Kratak sadržaj

Sadržaj

Nove karakteristike Delphija 5

Karakteristika	Poglavlje
Active Server Object Wizard i ASP podrška	20
ADO, DataSet komponente	12
Komponenta ApplicationEvents	6
Dodavanje debagera procesu koji se izvršava	18
Akcije i grupe tačke prekida	18
COM serveri u komponentama	16
Kontejnerske klase	4
Dijagrami podataka	10
Data Module Designer	10
FPU prozor	18
Okviri	4, 8
Procedura FreeAndNil	3
Integrated Translation Environment (ITE)	19
InterBase Express	11
Internet Express	21
Prečice menija	5
MIDAS 3	21
Nove klase TField	9
Poboljšanja Object Inspectora	1
Office komponente	16
Događaj OnContextPopup	5
Project Explorer	1
Poboljšanja Project Managera	1
Kategorije svojstava (registrovanje)	13
Resource Workshop	19
Sačuvana podešavanja radne površine	1
TeamSource	19
Todo List	1
Poboljšanja TypeInfoa	19
Windows kontrole, poboljšana podrška	7

Posetite sajt Kompjuter biblioteke - Delphi

Posetite naš sajt na adresi www.kombib.co.yu.

Preuzmite izvorni kod ove knjige sa adrese:

www.kombib.co.yu/Delphi5sourcecode.exe

Preuzmite dodatna originalna poglavlja sa adrese:

www.kombib.co.yu/knjige/preuzmite.htm

Saznajte najnovije informacije o Delphiju na adresi:

www.kombib.co.yu/zoinvrt/delphi.htm

Posetite Marcov Delphi web sajt za programere

Autor knjige, Marco Cantu, je načinio sajt namenjen Delphi programerima na adresi `www.marcocantu.com`. Sajt je odličan priručnik za sve Vaše Delphi programerske potrebe.

Sajt sadrži:

- Izvorni kod knjige (koji takođe možete pronaći na Sybex web sajtu)
- Dodatne primere i savete
- Veliki broj Delphi komponenata, čarobnjaka i alata koje je izradio autor
- Knjigu *Essential Pascal*
- Materijal o Delphiju koji se ne može pronaći u Helpu
- Neke dokumente koje je autor napisao o Delphiju, C++ i Javi
- Veliki broj linkova za Delphi web sajtove i dokumente
- Dokumente o autorovim knjigama, konferencije koje posećuje i njegove seminare

Sajt sadrži i novosti, sa posebnim delom koji se odnosi na autorove knjige, tako da čitaoci mogu da razmene mišljenje o sadržaju. Takođe, postoji i deo u kome se mogu razmeniti mišljenja o Delphi programiranju.

Posetite Sybexov sajt Mastering Delphi

Sybex web sajt, www.sybec.com, sadrži sve što Vam je potrebno da u potpunosti iskoristite *Delphi 5 Detaljan izvornik*. Kao što možete videti prelistavajući knjigu, Marco Cantu je načinio aplikacije i odgovarajući kod — više od 200 primera — koji će Vas voditi prilikom izrade aplikacija upotrebom Delphija 5.

SourceCode.exe je samoraspakujuća arhiva koja sadrži sav izvorni kod, uključujući .PAS fajlove, fajlove projekata i odgovarajuće DLL-ove. Svi čitaoci bi trebalo da preuzmu ovu arhivu da bi proučili izvorni kod i izradili programe.

Na web sajtu se nalazi i dodatno poglavlje nazvano **Graphics in Delphi** (Grafika u Delphiju). Ovo poglavlje sadrži:

- Crtanje na formularima
- Animirane kontrole
- Program za pregled slika
- Crtanje preko bitmapa
- Grafičke mreže i igre
- Upotrebu TeeCharta
- Windows metafajlove

Slobodno posetite web sajt knjige i preuzmite bilo šta što Vam je potrebno.

Evo kako možete preuzeti kod knjige i dodatno poglavlje:

1. Idite na `http://www.sybex.com`.
2. Kliknite Catalog.
3. Unesite **2565** (ISBN knjige) u polje za tekst.
4. Kliknite kontrolu Downloads na levoj strani ekrana.
5. Pročitajte Sybex Software License Agreement i prihvatite SYBEX User License. (Ukoliko ne želite da prihvatite, nećete moći da nastavite preuzimanje.)
6. Kliknite link za preuzimanje izvornog koda ili dodatnog poglavlja. (Možete preuzeti oba, ali jedan po jedan.)
7. Kada preuzmete kod, dva puta kliknite **sourcecode.exe**; zatim unesite direktorijum u koji želite da smestite fajlove, ili odaberite unapred određeni direktorijum (Md5code). Arhiva će se raspakovati u strukturu direktorijuma na osnovu delova i poglavlja knjige; na primer, `Part2\06\Borders` će sadržati izvorni kod za primer Borders Poglavlja 6.
8. Dodatno poglavlje je u Adobe Acrobat PDF formatu. Ukoliko koristite Netscape Navigator 3 ili bolju verziju, ili Microsoft Internet Explorer 3 ili bolju verziju, moći ćete da pogledate, sačuvate ili odštampate dokument iz pretraživača. Ukoliko Vam je potreban Acrobat Reader, možete otići na Adobe sajt sa *Mastering Delphi 5* Downloads strane.

Priznanja

Ovo izdanje knjige "Mastering Delphi" obeležava petu godinu ere Delphija. Kao i za mnoge druge programere, Delphi je bio u centru mog interesovanja tokom ovih godina, a pisanje, konsultacije, podučavanje i rad na konferencijama o Delphiju su zauzimale sve više i više mog vremena, ostavljajući ostale jezike i programske alate u prašini moje kancelarije. Kako su moj rad i život prilično isprepletani, mnogi ljudi su bili deo i jednog i drugog, i želeo bih da imam dovoljno prostora i vremena da im svima zahvalim za ono što zaslužuju. Umesto toga, ja ću pomenuti samo nekoliko ljudi i reći ću samo toplo: "Hvala" celom Delphi društvu — kao i za nagradu Spirit of Delphi 1999 Award koju sa ponosom delim sa Bobom Svartom (Bob Swart).

Prvu zvaničnu zahvalnost upućujem Borlandovim programerima i menadžerima koji su stvorili Delphi i koji su nastavili da ga unapređuju. To su Čak Jazdevski (Chuck Jazdzewski), Deni Torp (Danny Thorpe), Edi Čerčil (Eddie Churchill), Alen Bauer (Allen Bauer), Stiv Tod (Steve Todd), Mark Edington (Mark Edington), Džim Tirnej (Jim Tierney), Ravi Kumar (Ravi Kumar), Jerg Vejngarten (Jorg Weingaten), Anders Olson (Anders Ohlsson), i svi ostali koje nisam imao priliku da sretnem. Takođe, želim da pomenem moje prijatelje Bena Rigu (Ben Riga), koji je trenutno Delphi menadžer, Čarlija Kalverta (Charlie Calvert), Džona Kastera (John Kaster) i Dejvida I (David I). Ne mogu da zaboravim pomoć koju su mi pružili Zak Urloker (Zack Urlocker) i Nan Boreson (Nan Borreson).

Zatim sledi zahvalnost Sybexovom timu, u kojem su mnogi ljudi koje ne poznajem. Specijalnu zahvalnost upućujem Denis Santoro (Denise Santoro), Džimu Komptonu (Jim Compton) i Dajani Loveri (Diane Lowery) za njihov uvodnik; takođe želim da zahvalim Ričardu Milsu (Richard Mills), Kristini O'Kalahan (Kristine O'Callaghan), Morin Foris (Maureen Forys), Terezi Trego (Teresa Trego), Dženifer Kempbel (Jennifer Campbell), Kerol Iverson (Carol Iverson) i Toniju Joniku (Tony Jonick).

Ovo izdanje knjige "Mastering Delphi" imalo je neverovatan pregled koji je izvršio član Delphi tima Deni Torp (Denny Thorpe). Njegovi komentari su poboljšali knjigu u svim segmentima: tehničkom sadržaju, tačnosti, primerima, pa čak i u čitljivosti. Veliko hvala. I u prethodnim izdanjima bilo je dosta doprinosa sa strane: Tim Guč (Tim Gooch) je radio na Delu V knjige "Mastering Delphi 4", a Đuzepe Madafari (Giuseppe Madaffari) je dao veliki doprinos materijalu koji se odnosi na baze podataka u ovom i prethodnom izdanju. Mnoga poboljšanja tekstu i primerima su doneli tehnički revizori prethodnih izdanja — Huankarlo Anjez (Juancarlo Anez), Ralf Fridman (Ralph Friedman), Tim Guč (Tim Gooch) i Alan Tadros (Alain Tadros) — kao i revizije koje su ranije učinili Bob Svart (Bob Swart), Đuzepe Madafari (Giuseppe Madaffari) i Stiv Tendon (Steve Tendon).

Naročito zahvaljujem mojim prijateljima Brusu Ekelu (Bruce Eckel), Andrei Provaljio (Andrea Provaglio), Normu Mekintošu (Norm McIntosh) Džoani (Johanna) i Filu (Phil) iz BUG-UK, Reju Konopki (Ray Konopka), Marku Mileru (Mark Miller), Keriju Jensenu (Cary Jensen), Krisu Frizeli (Chris Frizelle) iz The Delphi Magazina, Fu Sej Hauu (Foo Say How), Džonu Houvu (Jonh Howe), Majku Orisu (Mike Orriss), Čadu "Kudzu" Haueru (Chad "Kudzu" Hower), Denu Miseru (Den Miser) i Marku Miotiju (Marco Miotti). Takođe, veoma veliko hvala i svim polaznicima

mojih programerskih kurseva o Delphiju, posetiocima mojih seminara i konferencija u Italiji, SAD, Francuskoj, Velikoj Britaniji, Singapuru, Holandiji, Nemačkoj i Švedskoj.

Pored ljudi koji imaju udela u Delphiju, najveću zahvalnost šaljem svojoj strpljivoj supruzi Leli (Lella), koja je (dok je bila u drugom stanju) morala da provede još jedno leto sa veoma malo odmora, jer knjiga uvek odnosi više vremena nego što ja to očekujem. Mnogi naši prijatelji su nam omogućili kratke odmore tokom rada: Sandro (Sandro) i Monika (Monica) sa Lukom (Luca), Stefano (Stefano) i Elena (Elena), Marko (Marco) i Lora (Laura) sa Mateom (Matteo), Bjankom (Bianca), Kjarom (Chiara), Lukom (Luca) i Elenom (Elena), Kjara (Chiara) i Daniel (Daniele) sa Leonardom (Leonardo), Lorom (Laura), Vito (Vito) i Marika (Marika) sa Sofijom (Sofia). Naši roditelji, braća, sestre i njihove porodice su dali veliku podršku. Bilo je lepo provesti nešto našeg slobodnog vremena sa njima i naših šestoro sestrića, Mateom (Matteo), Andreom (Andrea), Đakomom (Giacomo), Stefanom (Stefano), Andreom (Andrea) i Pjetrom (Pietro).

Na kraju, želeo bih da zahvalim svim ljudima, od kojih mnoge i ne poznajem, koji uživaju u životu i pomažu da bi izgradili bolji svet. Ukoliko nikada ne prestanem da verujem u budućnost i mir, to će biti zahvaljujući njima.

Marco Cantu

Uvod

Prvi put kada mi je Zak Urloker (Zack Urlocker) pokazao proizvod kodiranog naziva Delphi koji je trebalo prezentovati, ja sam shvatio da će to promeniti moj rad — i rad mnogih drugih programera. Ja sam se nekada borio sa C++ bibliotekama za Windows, a Delphi je bio i ostao najbolja kombinacija objektno orijentisanog programiranja i vizuelnog programiranja za Windows.

Delphi 5 jednostavno nastavlja ovu tradiciju i solidne osnove VCL-a da bi se dobio još jedan neverovatan i sveobuhvatan softverski alat za programiranje. Potrebna su Vam rešenja za baze podataka, klijent/server, višelinijsko, intranet ili Internet programiranje? Potrebne su Vam kontrola i snaga? Potreban Vam je brz razvoj? Uz Delphi 5 i veliki broj tehnika i saveta koji se nalaze u ovoj knjizi, moći ćete sve to da ostvarite.

Pet verzija, a usavršavanje se nastavlja

Neke od prvobitnih karakteristika Delphija koje su me privukle su bili pristup na osnovu formulara i objektno orijentisani pristup, neverovatno brz kompajler, velika podrška za baze podataka, dobra integracija sa Windows programiranjem i tehnologija komponenata. Međutim, najvažniji element je bio jezik Object Pascal koji je osnova za sve ostalo.

Delphi 2 je čak bio i bolji! Među svim poboljšanjima su bila i sledeća: Multi-Record Object i poboljšane tabele za baze podataka, podrška za OLE Automation i variant tip podataka, potpuna podrška i integracija za Windows 95, long string tip podataka i Visual Form Inheritance. Delphi 3 je ovome dodao tehnologiju Code Insight, podršku za DLL debagovanje, šablone komponenata, TreeChart, Decision Cube, Web Broker tehnologiju, pakete komponenata, ActiveForms i neverovatnu integraciju sa COM-om, zahvaljujući interfejsima.

Delphi 4 nam je podario AppBrowser editor, nove karakteristike Windowsa 98, poboljšanu podršku za OLE i COM, proširene komponente za baze podataka i mnoge dodatke osnovnim VCL klasama, uključujući podršku za dokiranje, povezivanje i usidrenje kontrola. U Delphiju 4 postoji mnogo novih karakteristika, koje možete otkriti čitajući ovu knjigu, ukoliko ste propustili prethodno izdanje.

Delphi 5 sveukupnom utisku dodaje još više poboljšanja IDE-a (previše da bismo ih ovde nabrojali), proširenu podršku za baze podataka (sa specifičnim ADO i InterBase skupovima podataka), poboljšanu verziju MIDAS-a sa podrškom za Internet, TeamSource kontrolu verzija, mogućnosti prevođenja, koncept okvira, mnogo novih komponenata i još mnogo toga kao što ćete videti na stranama ove knjige.

Delphi je izvrstan alat, ali je, takođe, i složeno programersko okruženje koje sadrži veliki broj elemenata. Ova knjiga će Vam pomoći da u potpunosti savladate Delphi programiranje, uključujući i jezik Object Pascal, Delphi komponente (kako upotrebu postojećih tako i izradu sopstvenih), podršku za baze podataka i klijent/server podršku, ključne elemente COM i Windows programiranja i Internet i Web programiranje.

Nije Vam potrebno veliko znanje bilo koje teme da biste pročitali ovu knjigu, ali je potrebno da znate osnove Pascal programiranja. Ukoliko ste već upoznali Delphi, to će Vam mnogo pomoći, naročito posle nekoliko uvodnih poglavlja. Knjiga od samog početka detaljno opisuje teme; veći deo uvodnog materijala iz prethodnih izdanja je izostavljen. Nešto od izostavljenog materijala i uvoda u Pascal se može naći na autorovom web sajtu i može poslužiti kao polazna tačka ukoliko niste upoznati sa osnovama Delphija. Svaka nova karakteristika Delphija 5 je objašnjena u odgovarajućim poglavljima u ovoj knjizi.

Struktura knjige

Knjiga je podeljena u pet delova:

- Deo I, "Delphi i objektni Pascal", predstavlja nove karakteristike integrisanog razvojnog okruženja Delphija 5 (Integrated Development Environment — IDE) u Poglavlju 1, a zatim se prelazi na jezik Object Pascal i Visual Component Library (VCL) i daju se osnove i saveti.
- Deo II, "Upotreba komponenata", odnosi se na standardne komponente, Windows uobičajene kontrole, grafiku, menije, okvire za dijalog, skrolovanje, dokiranje, kontrole sa više strana, Multiple Document Interface i mnogo toga drugog.
- Deo III, "Pisanje aplikacija za baze podataka", opisuje pristup bazama podataka, napredne Paradox teme, detaljno opisuje komponente koje prepoznaju podatke, klijent/server programiranje, InterBase Express i ADO.
- Deo IV, "Komponente i biblioteke", opisuje Delphi komponente i programiranje dinamičke biblioteke za povezivanje (Dynamic Link Library — DLL); zatim se prelazi na COM i OLE, opisuju se proširenja Windows školjke, OLE Automation i ActiveX programiranje.
- Deo V, "Praktično Delphi programiranje", razmatra mnoge uobičajene programerske tehnike, kao što su rukovanje memorijom, debagovanje, upotreba resursa, podrška štampanju, rukovanje fajlovima, programiranje TCP/IP priključaka, Internet programiranje, web ekstenzije na strani servera, višelinijska arhitektura i distribuirane aplikacije za baze podataka koje se izrađuju na osnovu MIDAS tehnologije

Kao što pokazuje ovaj kratak rezime, ova knjiga obuhvata sve teme koje korisnici Delphija imaju na svim nivoima programerskog znanja, od "naprednih početnika" do programera komponenata.

U ovoj knjizi sam pokušao da u potpunosti izostavim referentni materijal i da se umesto toga usredsredim na tehnike efektivne upotrebe Delphija. Kako Delphi obezbeđuje detaljnu dokumentaciju, prikazivanje svih metoda i svojstava u ovoj knjizi ne bi bilo dobro, a knjiga bi zastarela čim bi se softver malo promenio. Savetujem Vam da dok čitate ovu knjigu, pri ruci imate Delphi Help fajlove tako da Vam referentni materijal bude na raspolaganju. Više referentnog Delphi materijala možete pronaći na mom web sajtu, što će kasnije biti opisano.

Bilo kako bilo, ja sam se potrudio da Vam omogućim čitanje knjige deleko od kompjutera, ukoliko Vam to više odgovara. Slike ekrana i ključni delovi listinga bi trebalo da Vam pomognu. U knjizi se koristi nekoliko konvencija da bi bila čitljivija. Svi elementi izvornog koda, kao što su ključne reči, nazivi svojstava, klase i funkcije su prikazani u `ovom fontu`, a listinzi su formatirani onako kako se pojavljuju u Delphi editoru, kada su ključne reči prikazane masnim slovima, a komentari kurzivom.

Besplatni izvorni kod na Webu

Ova knjiga se usredsređuje na primere. Posle prikaza svakog koncepta ili Delphi komponente pronaći ćete program koji funkcioniše (ponekad i više od jednog) i koji pokazuje kako se može upotrebiti. Sve je prikazano i postoji više od 200 primera u knjizi. Ovi programi su direktno dostupni kako na web sajtu izdavača (`www.sybex.com`) tako i na autorovom web sajtu (`www.marcocantu.com`, gde takođe možete pronaći ažurirane primere iz prethodnih izdanja). U uvodu knjige možete pronaći potpuna uputstva o preuzimanju i instaliranu samoraspakujućih arhiva koje sadrže softver, bilo sa sajta Sybexa ili sa našeg sajta. Potrebno je da programe preuzmete pre nego što počnete rad sa primerima knjige. Većina primera je veoma jednostavna i odnosi se na samo jednu karakteristiku. Složeniji primeri su često izrađeni korak po korak, sa međukoracima koji uključuju delimična rešenja i slojevita poboljšanja.

NAPOMENA

Neki primeri baza podataka zahtevaju da imate Delphi primer baze podataka DBDEMOS; to je deo unapred određene Delphi instalacije. ■

Pored arhive koja sadrži minimalne fajlove sa izvornim kodom koji su neophodni za izradu programa, druga arhiva sadrži HTML verziju izvornog koda u kojoj je istaknuta sintaksa, sa potpunim prikazom ključnih reči i identifikatora (klasa, funkcija, metoda, naziva svojstava). Ovakav prikaz se nalazi u HTML fajlu tako da lako možete da koristite svoj pretraživač da biste pronašli sve programe koji koriste Delphi ključnu reč ili identifikator koji tražite.

Struktura direktorijuma preuzetih fajlova je prilično jednostavna. U osnovi, svaki deo knjige se nalazi u zasebnom direktorijumu, sa poddirektorijumima za svako poglavlje i sa poddirektorijumima za svaki primer (recimo, `Part2\06\Borders`). U tekstu se na primere referiše nazivom (recimo, Borders).

NAPOMENA

Obavezno pročitajte fajl nazvan Readme koji se nalazi u arhivi i koji sadrži važne informacije o legalnoj i efikasnoj upotrebi softvera. ■

Kako stupiti u vezu sa autorom

Ukoliko imate bilo kakve probleme sa tekstom ili primerima knjige, i izdavač i ja ćemo biti na raspolaganju. Pored prijavljivanja grešaka i problema, molimo Vas da nam pošaljete iskreno mišljenje o knjizi i da nam kažete koji primeri su Vam najviše pomogli, a koji Vam se najviše dopadaju. Postoji nekoliko načina na koje nas možete informisati:

- Na Sybexovom web sajtu (`www.sybex.com`) možete pronaći izvorni kod knjige kao i ažurirani tekst ukoliko je potrebno. Da biste poslali komentar knjige, kliknite Contact, a zatim odaberite Book Content Issues. Ovaj link će Vam prikazati gde možete da unesete svoje komentare.
- Na mojoj web strani (`http://www.marcocantu.com`) nalaze se novosti i saveti, članci i besplatna knjiga Essential Pascal, referentne informacije o Delphiju 5 koje nisu stale u ovu knjigu i moja kolekcija komentara Delphi komponenata i alata.
- Ja sam, takođe, svojim knjigama posvetio grupu koja se nalazi na adresi `news://news.marcocantu.it/public`. Ostale oblasti na adresi `news://news.marcocantu.it/public` (deo na engleskom) su posvećene pitanjima o Delphiju i drugim temama. Pogledajte moj web sajt da biste videli oblasti i da biste se prijavili. (Zapravo, sve ovo je besplatno, ali morate imati lozinku).
- Na kraju, možete stupiti u vezu sa mnom putem elektronske pošte (`marco@marcocantu.com`). Molim Vas da za tehnička pitanja prvo iskoristite grupu, jer ćete možda brže dobiti odgovor. Moje poštansko sanduče je često prepuno te ne mogu odmah da odgovorim svakom zahtevu. Molim Vas da mi pišete na engleskom ili italijanskom.

I

Delphi 5 i Object Pascal

U OVOM DELU:

1. Delphi 5 - Integrisano razvojno okruženje
2. Objektno orijentisano programiranje u Delphiju
3. Unapređeni Object Pascal
4. VCL tehnike programiranja

POGLAVLJE

1

Delphi 5 integrisano razvojno okruženje

U vizuelnom programskom alatu kao što je Delphi, uloga okruženja je čak važnija od samog programskog jezika. Delphi 5 nudi mnoge nove mogućnosti u svom vizuelnom razvojnom okruženju i u ovom poglavlju ćemo ih sve prikazati. Ovo poglavlje nije potpuno uputstvo već, uglavnom, kolekcija saveta i sugestija namenjena prosečnom korisniku Delphija. Drugim rečima, poglavlje nije namenjeno novim korisnicima. Ja ću se pozabaviti novim mogućnostima integrisanog razvojnog okruženja Delphija 5 (Integrated Development Environment – IDE) i nekim naprednim i/ili manje poznatim mogućnostima prethodnih verzija, ali u ovom poglavlju neću obezbediti instrukcije korak-po-korak. U knjizi ću podrazumevati da znate kako da sprovedete osnovne IDE operacije, i sva poglavlja posle ovog će se odnositi isključivo na programerske tehnike.

Ukoliko ste programer početnik, nemojte da se plašite. Integrisano razvojno okruženje Delphi je prilično intuitivno. Delphi sadrži uputstvo (možete ga pronaći u formatu Acrobat na Delphi CD-u) sa delom koji predstavlja razvoj Delphi aplikacija. Takođe, možete pronaći instrukcije korak-po-korak za Delphi IDE na mom web sajtu `www.marcocantu.com`. Kratka knjiga *Essential Delphi* je zasnovana na materijalu prvih poglavlja prethodnih izdanja knjige *Mastering Delphi*.

Izdanja Delphija 5

Pre nego što uronimo u detalje programskog okruženja Delphi, posvetimo pažnju dvema ključnim idejama. Prvo, ne postoji jedno izdanje Delphija 5; postoje mnoga izdanja. Drugo, svako Delphi okruženje se može prilagoditi. Zbog toga se Delphi ekrani, koje ćete videti u ovom poglavlju, mogu razlikovati od ekrana na Vašem kompjuteru. Evo aktuelnih izdanja Delphija.

- Osnovna verzija (izdanje "Standard") je namenjena novim korisnicima Delphija i povremenim programerima.
- Drugi nivo (izdanje "Professional") je namenjen profesionalnim programerima. Sadrži sve osnovne mogućnosti i podršku za programiranje baza podataka, obimnu podršku web serverima (WebBroker) i neke dodatne alate. Knjiga podrazumeva da koristite bar verziju Professional.
- Celokupni Delphi (izdanje "Enterprise", ranije označeno kao "Client/Server Suite") je namenjen programerima koji izrađuju velike aplikacije. Sadrži SQL Links za osnovne konekcije Client/Server BDE, komponente ADO i Intrebase Express, podršku za višekorisničke aplikacije, internacionalnu podršku, arhitekturu drveta i mnoge druge alate, uključujući i SQL Monitor. Neka poglavlja se odnose samo na mogućnosti koje su dostupne u izdanju Delphi Enterprise; ti odeljci su jasno označeni.

Neke mogućnosti izdanja Delphi Enterprise vlasnici izdanja Delphi Professional mogu dobiti dokupljivanjem. Mada je to komercijalna odluka koja se može promeniti u budućnosti, trebalo bi da možete da nabavite komponente ADO i TeamSource (služi za uporedni rad više programera). Ukoliko ne možete da opravdate cenu punog izdanja Enterprise, možda možete da kupite Delphi Professional i određene podsisteme koje želite od Borland Online Storea.

Pored toga što su dostupna različita izdanja, postoje i brojni načini prilagođavanja Delphi okruženja. U ilustracijama ekrana kroz knjigu pokušao sam da koristim standardni korisnički interfejs (onako kako izgleda posle instalacije); ipak, ja imam neke sklonosti, naravno, i instaliram mnoge dodatke koji mogu uticati na neke prikaze ekrana.

Delphi 5 IDE

Delphi 5 IDE sadrži neke velike izmene koje je Borland predstavio od napretka Delphija 1 u Delphiju 2. Među novim funkcijama su i redizajnirani Object Inspector, novi Project Manager, mogućnost čuvanja pozicije prozora, spisak šta treba uraditi i još mnogo toga. Mnoge funkcije je lako razumeti, ali ih vredi pažljivo proučiti da biste mogli da u potpunosti koristite Delphi 5.

Opcije komandne linije

Prva stvar koju treba imati na umu je da postoje izmene i pre nego što pokrenete Delphi. Zapravo, program delphi32.exe, koji pokreće IDE, ima mnoge opcije komandne linije. Većina ovih opcija (prikazane su u temi Help IDE command-line options) je namenjena iskusnijim korisnicima i omogućava praćenje statusa Delphi IDE-a.

Na primer, možete učitati program u debager ili pridružiti sistem procesu koji se već izvršava (teme koje ću razmatrati uz ostale funkcije debagovanja u Poglavlju 18).

Ostale funkcije mogu da budu korisne čak i povremenim programerima. Zastavica `-ns` (no splash flag) preskače početni ekran, a zastavica `-np` (no project flag) govori Delphiju da prilikom pokretanja ne otvara prazan (novi) projekat. (Ovim se omogućava brzo pokretanje jer se sprečava učitavanje bilo kog paketa komponenata koji je pridodat projektima.)

Verovatno najčešće korišćena funkcija, nije striktno opcija komandne linije ili čak opcija prilikom pokretanja. Lako možete naznačiti projekat, grupu projekta ili fajl izvornim kodom Pascal koji treba otvoriti. Kada se Delphi već izvršava, neće se otvoriti nova kopija IDE-a kada dva puta kliknete naziv fajla ili ikonu u Windows Exploreru. Jednostavno će se otvoriti PAS ili DFM fajl u aktuelnoj kopiji Delphija. Kada označite fajl projekta (.DPR), Delphi prvo zatvara aktuelni projekat pošto Vas upita da li da sačuva izmene.

Sa komandne linije možete učitati projekat i dozvoliti Delphiju da automatski izradi ili načini projekat (upotrebom opcija `-b` i `-m`), zatvarajući IDE posle kompletiranja opcije. Ovo ne izgleda naročito korisno; za kompajliranje niza velikih projekata sa skript ili batch fajlom bolje je da koristite brži kompajler sa komandnom linijom (kojem nije potreban IDE).

Čuvanje izgleda radne površine

Na osnovama ranijih verzija Delphija i na podršci stalnim pozicijama koje su dodate u Win32, Delphi je od verzije 4 omogućio programerima da prilagode IDE na brojne načine kada se otvara mnogo prozora i kada se prozori pomeraju i priključuju jedni drugima. Ipak, programerima je često potrebno da otvore jedan skup prozora prilikom dizajniranja a drugi skup prozora prilikom debagovanja. Slično tome, programerima je, možda, potreban jedan izgled kada rade sa formularima, a potpuno drugačiji kada izrađuju komponente ili kod niskog nivoa kada koriste samo editor. Preuređenje IDE-a za svaki od ovih zadataka je dosadan posao.

U Delphiju 5, svaki put kada načinite uređenje IDE prozora za neku specifičnu namenu, možete da ga sačuvate pod nekim nazivom i lako ga uspostavite. Takođe, možete neko od ovih uređenja načiniti osnovnim izgledom prilikom debagovanja tako da se automatski uspostavlja kada pokrenete debager. Sve ove funkcije su dostupne sa nove palete alata Desktops prikazane na slici 1.1. (To je jedina paleta alata na kojoj je prikazano combo polje.) Takođe, možete raditi sa izgledom radne površine koristeći meni View➥Desktops. Ovaj meni sadrži funkcije palete alata, a takođe Vam omogućava da uklonite neko od sačuvanih uređenja.

SLIKA 1.1 *Glavni prozor Delphija 5 sadrži paletu alata Desktops, koju možete korisiti da ponovo učitate uređenje IDE prozora*

Informacije o izgledu radne površine se čuvaju u DST fajlovima, koji su u osnovi INI fajlovi. Sačuvana uređenja sadrže poziciju glavnog prozora, Project Manager, Alignment Palette, Object Inspector (uključujući nove vrednosti svojstava kategorije), prozore editora (sa statusom Code Explorera i Message Viewa) i mnoge druge, kao i status priljubljivanja raznih prozora.

Evo malog dela DST fajla koji bi trebalo da je lako razumljiv:

```
[Main Window]
Create=1
Visible=1
State=0
Left=0
Top=0
Width=1024
Height=105
ClientWidth=1016
ClientHeight=78

[ProjectManager]
Create=1
Visible=0
State=0
…
Dockable=1

[AlignmentPalette]
Create=1
Vislble=0
…

[PropertyInspector]
Create=1
Visible=1
…
Dockable=1
SplitPos=85
ArrangeBy=Name
HiddenCategories=Legacy
ShowStatusBar=1
```

NAPOMENA

Uređenje radne površine zanemaruje uređenje projekta. Ovim se prevazilazi problem prebacivanja projekta sa mašine na mašinu (ili između programera) i problem preuređenja prozora prema sklonostima. Delphi 5 odvaja sklonosti prema korisniku i prema mašini od uređenja projekta da bi bolje podržao timski razvoj. ■

Spisak stvari koje treba uraditi

Još jedna potpuno nova funkcija Delphi 5 IDE-a je spisak stvari koje treba uraditi. To je spisak zadataka koje treba da uradite da biste kompletirali projekat, skup beležaka za programera (ili programere, jer ovaj alat može da bude veoma koristan u timskom radu). Mada ideja nije nova, ključni koncept spiska u Delphiju 5 je to što se spisak ponaša kao dvosmerni alat.

Zapravo, možete dodavati ili menjati elemente spiska dodavanjem specijalnih komentara izvornom kodu bilo kog fajla projekta; zatim ćete videti odgovarajuće stavke u listi. Međutim, možete i vizuelno menjati elemente liste da biste modifikovali odgovarajuće komentare izvornog koda. Na primer, evo kako element spiska može izgledati u izvornom kodu:

```
procedure TForm1. FormCreate(Sender: TObject);
begin
  // TODO -oMarco: Add creation code
end;
```

Isti element se može menjati u prozoru prikazanom na slici 1.2.

SLIKA 1.2 *Prozor Edit To-Do Item se može koristiti za izmene elementa spiska; operacija koja se može izvesti direktno u izvornom kodu*

Izuzetak od ovog dvosmernog pravila je definicija elemenata spiska koji se odnose na ceo projekat. Takve elemente morate uneti direktno u spisak. Da biste to učinili, možete upotrebiti kombinaciju tastera Ctrl+A u prozoru To-Do List ili možete kliknuti desnim tasterom miša u prozoru i odabrati Add iz kontekst menija. Ovi elementi se čuvaju u posebnom fajlu čija je ekstenzija .TODO.

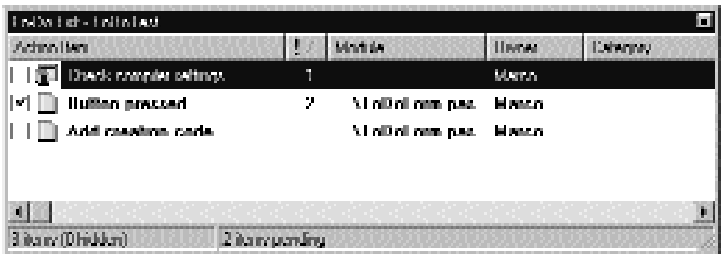
Postoji više opcija koje možete upotrebiti uz komentar TODO. Možete koristiti `-o` (kao u delu koda iznad) da biste naznačili vlasnika, programera koji je uneo komentar; opciju `-c` da biste naznačili kategoriju, ili jednostavno broj između 1 i 5 da biste naznačili prioritet (0 ili ukoliko nema broja, znači da nije određen nivo važnosti). Na primer, upotrebom komande Add To-Do Item iz kontekst menija editora (ili tastaturne prečice Ctrl+Shift+T) generiše se ovakav komentar:

```
{ TODO 2 -oMarco : Button pressed }
```

Delphi tretira sve iza zareza, sve do kraja reda ili zatvorene zagrade, prema tipu komentara, kao tekst elementa spiska.

Konačno, u prozoru To-Do List možete potvrditi element da biste naznačili da je zadatak urađen. Komentar izvornog koda će se promeniti iz `TODO` u `DONE`. Takođe, možete ručno promeniti komentar u izvornom kodu da biste videli oznaku u prozoru To-Do List.

Jedan od najmoćnijih elemenata ovakve arhitekture je glavni prozor To-Do List, prikazan na slici 1.3, koji automatski može da prikupi informacije iz fajlova sa izvornim kodom dok ih unosite. Elementi ovog spiska su deo primera ToDoTest ovog poglavlja (koji ne rade ništa osim što predstavljaju spisak zadataka koje treba obaviti). Elementi spiska u ovom prozoru prikazuju različite atribute koje sam upravo opisao, kao i fajlove izvornog koda u kojima su definisani. Inicijalno polje za potvrdu je označeno za elemente Done, a njihov tekst je precrtan.

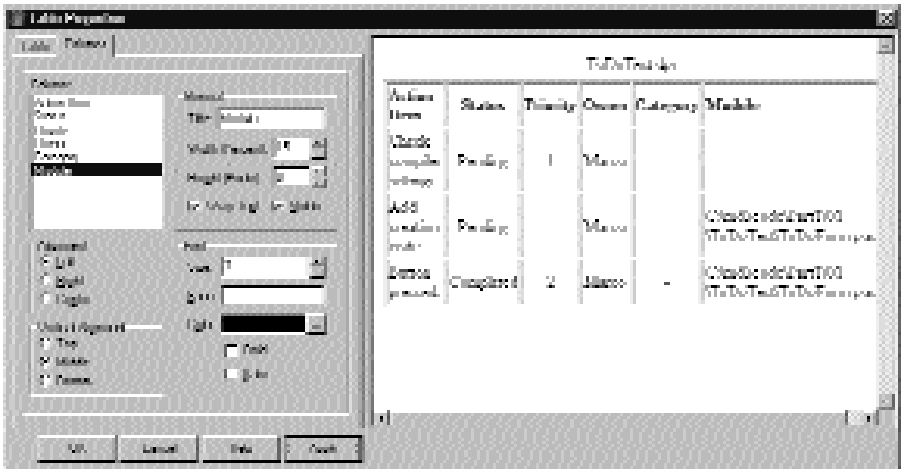


SLIKA 1.3 *Prozor To-Do za primer ToDoTest*

NAPOMENA

Da biste isprobali ToDoTest i sve primere programa ove knjige, potrebno je da preuzmete izvorni kod sa web sajta Sybex. (Pronaći ćete kompletne instrukcije za preuzimanje na unutršnjoj strani korica.) Svaki čitalac bi trebalo da preuzme izvorni kod, da bi u potpunosti iskoristio ovu knjigu. Svaki put kada u tekstu naiđete na na novi program koji se navodi nazivom, potrebno je da potražite direktorijum takvog naziva u preuzetim fajlovima i pročitate kompletan izvorni kod. Za većinu primera ćete, takođe, želeti da kompajlirate program i pokrenete ga. ■

Prozor To-Do List sadrži kontekst meni koji Vam omogućava da dodate, izmenite ili uklonite elemente, izdvojite ih i sortirate, i izvezete na Clipboard. Komanda koja se koristi da bi se obavila poslednja operacija, Copy As, Vam omogućava da izvezete elemente bilo kao tekst bilo kao tabelu HTML, koju možete prilagoditi upotrebom komande Table Properties. HTML podešavanja tabele uključuju i preliminarni prikaz, kao što možete videti na slici 1.4. Informacije se ne čuvaju u HTML fajlu; samo se kopiraju na Clipboard. Potrebno je da pokrenete Vaš omiljeni HTML editor (ili Notepad ili tekst-prozor u editoru Delphija) da biste sačuvali fajl.



SLIKA 1.4 *HTML preliminarni prikaz tabele spiska stvari koje treba uraditi*

Editor AppBrowser

Editor koji se nalazi u Delphiju 5 se nije mnogo promenio od Delphija 4. Ipak, Delphi 4 je sadržao mnogo novih funkcija tako da nije na odmet kratko objasniti ovaj alat. Delphi 4 je doneo tri osnovne inovacije: prozor Code Explorer (koji prikazuje sve definicije jedinica), podršku navigaciji (slično web pretraživačima) i Class Completion (tehnologiju generisanja koda).

Delphi 5 dodaje editoru novo mapiranje tastature za emulaciju Visual Studia i mogućnost proširivanja editora korisničkim modulima mapiranja tastature. Ova poslednja podešavanja su definisana pod jezičkom Key Bindings okvira za dijalog Editor Properties, koji možete aktivirati komandom Tools➥Editor Options. Ovaj novi okvir za dijalog prikazuje podešavanja okruženja koja se tiču editora.

NAPOMENA

Korisnički moduli mapiranja tastature se mogu napisati upotrebom novih funkcija Tools API koje su dodate Delphiju 5. Možete napisati potpuno novi modul za mapiranje tastature ili dodati nekoliko posebnih tastaturnih prečica postojećim. Napredne teme nećemo razmatrati u knjizi, ali možete pronaći primere u direktorijumu Editor Keybinding direktorijuma Delphi Demos. Jedno od ovih dodatnih ograničenja, koje se naziva Buffer List, se instalira po definiciji i dostupno je kada upotrebite kombinaciju tastera Ctrl+B. ■

Delphi editor Vam omogućava da radite na više fajlova odjednom, koristeći metaforu "beležnice sa više jezičaka", a možete otvoriti i više prozora editora. Možete da pređete sa jedne strane editora na drugu, ukoliko pritisnete kombinaciju tastera Ctrl+Tab (ili Shift+Ctrl+Tab da biste se kretali u suprotnom smeru). Postoji veliki broj opcija koje utiču na editor, a nalaze se u novom okviru za dijalog Editor Properties. Potrebno je, ipak, da pređete na stranu Preferences okvira za dijalog Environment Options, da biste podesili funkciju AutoSave koja čuva fajlove sa izvornim kodom, svaki put kada pokrenete program (sprečavajući gubitak podataka ukoliko se program zaglavi).

Neću razmatrati razna podešavanja editora jer su prilično intuitivna i opisana su u Helpu. Ono što zvanično nije dokumentovano, je to da možete upotrebiti dve stavke Windows Registryja da biste podesili početnu širinu i visinu editora (da bi bio veliki koliko i ekran, na primer). Pronađite sekciju Delphi u Registryju, `HKEY_CURRENT_USER/Software/Borland/Delphi/5.0`, i pod ključem `Editor` dodajte dva nova elementa `DWORD`, nazovite ih `DefaultHeight` i DefaultWidth, koji označavaju visinu i širinu editora u pikselima. Da biste izmenili Windows Registry, možete upotrebiti aplikaciju RegEdit.EXE pod Windowsom 95 i 98 ili RegEdt32.EXE pod NT-om.

Još jedan savet, koji treba zapamtiti, je da počevši od Delphija 4, komande Cut i Paste nisu jedini način za premeštanje izvornog koda. Takođe, možete označiti i prevući reči, izraze ili cele redove izvornog koda. Možete kopirati tekst umesto da ga premestite, tako što ćete pritisnuti i držati pritisnut taster Ctrl prilikom prevlačenja.

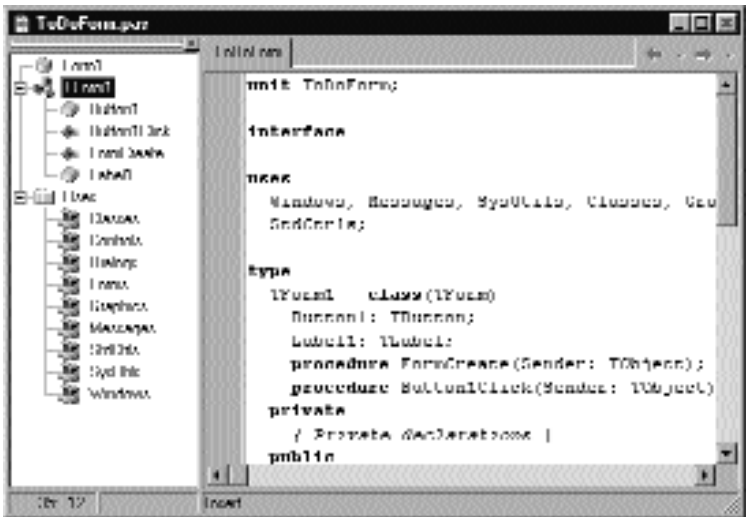
Code Explorer

Prozor *Code Explorer*, koji je, uopšte uzev, najkorisniji kada je priljubljen uz stranu editora, jednostavno prikazuje sve tipove, promenljive i rutine definisane u okviru jedinice, i još i druge jedinice koje se prikazuju u iskazima 'uses'. Za kompleksne tipove, kao što su klase, Code Explorer može prikazati spisak detaljnih informacija uključujući i spisak polja, svojstava i metoda. Sve informacije se ažuriraju čim počenete da unosite u prozor editora. Možete upotrebiti Code Explorer da biste se kretali kroz editor. Ukoliko dva puta kliknete na neku stavku u Code Exploreru, editor prelazi na odgovarajuću deklaraciju.

Mada je sve ovo očigledno posle nekoliko minuta korišćenja Delphija, postoje neke funkcije Code Explorera koje nisu tako intuitivne. Jedan važan aspekat je da imate potpunu kontrolu nad izgledom informacija i možete ograničiti dubinu drveta koje se obično prikazuje u ovom prozoru podešavanjem Code Explorera. Smanjivanje drveta može pomoći bržem označavanju. Možete konfigurisati Code Explorer upotrebom odgovarajuće strane okvira za dijalog Environment Options, kao što se vidi na slici 1.5.

SLIKA 1.5 *Možete konfigurisati Code Explorer u okviru za dijalog Evironment Options*

Primetićete da, kada poništite neki od elemenata Explorer Categories na desnoj strani okvira za dijalog, Explorer ne uklanja odgovarajuće elemente iz pogleda. On jednostavno dodaje čvor na drvo. Na primer, ukoliko poništite Uses, Delphi ne sakriva spisak upotrebljenih jedinica iz Code Explorera. Suprotno, upotrebljene jedinice su prikazane u spisku kao glavni čvorovi, a ne kao direktorijum Uses. Kao još jedan primer, onemogućavanjem sekcija Types, Classes i Variables dobijate prikaz koji možete videti na slici 1.6.



SLIKA 1.6 *Neki direktorijumi Code Explorera se mogu ukloniti uklanjanjem elemenata odgovarajućih podešavanja*

Najvažnija podešavanja su najverovatnije ona koja se odnose na klase. Definicije koje se odnose na klase se mogu urediti na tri načina:

- prema kategorijama private, protected, public i published;
- prema metodima i poljima kategorija;
- sve zajedno kao jedna grupa.

Kako je svaki element drveta Code Explorer označen ikonom koja identifikuje njegov tip, uređenje po polju i metodu izgleda manje važno nego uređenje prema specifikatoru pristupa. Ja više volim da sve elemente prikažem u jednoj grupi, jer to zahteva najmanju upotrebu miša da bi se pristupilo nekom elementu. Označavanje elemenata u Code Exploreru, zapravo, obezbeđuje zgodan način kretanja kroz izvorni kod velike jedinice. Kada dva puta kliknete metod u Code Exploreru, prelazi se na definiciju u deklaraciji klase (u interfejs delu jedinice). Možete upotrebiti kombinaciju tastera Ctrl+Shift i kursor tastera nagore ili nadole da biste prešli sa definicije metoda ili procedure iz interfejsa jedinice na potpunu definiciju u delu implementacije (ili ponovo natrag).

NAPOMENA

Neke od Explorer Categories prikazanih na slici 1.5 se koriste u novom Project Exploreru (ili Browseru) koji je predstavljen u Delphiju 5, a ne u Code Exploreru. To se odnosi, između ostalog, na opcije grupisanja Virtuals, Statics, Inherited i Introduced. ■

Code Explorer nije samo alat za izdavanje i pretraživanje. Zapravo, možete ga koristiti za unošenje novih elemenata za svaku od kategorija. Tip novog elementa uopšteno zavisi od onoga što unesete. Naziv koji počinje ključnom reči `procedure` ili `function` se automatski smatra metodom, dok naziv za kojim sledi tačka-zarez i tip podataka se smatra poljem. Mogućnosti izmena Code Explorera su previše ograničene da bi obezbedile značajnu prednost nad editovanjem u prozoru izvornog koda. Bilo bi lepo imati mogućnost prevlačenja, na primer, da bi se premestilo polje ili metod u neku drugu sekciju ili kopirati u neku drugu klasu.

NAPOMENA

Polje, metodi, javno, privatno…? Ukoliko niste upoznati sa terminologijom jezika Object Pascal, u Poglavlju 2 ćete pronaći prilično dobro objašnjenje ovih termina. Ja sam ih tamo navodio bez nekog naročitog objašnjenja jer je većina čitalaca ove knjige verovatno nekada koristila Delphi i njegov programski jezik. ■

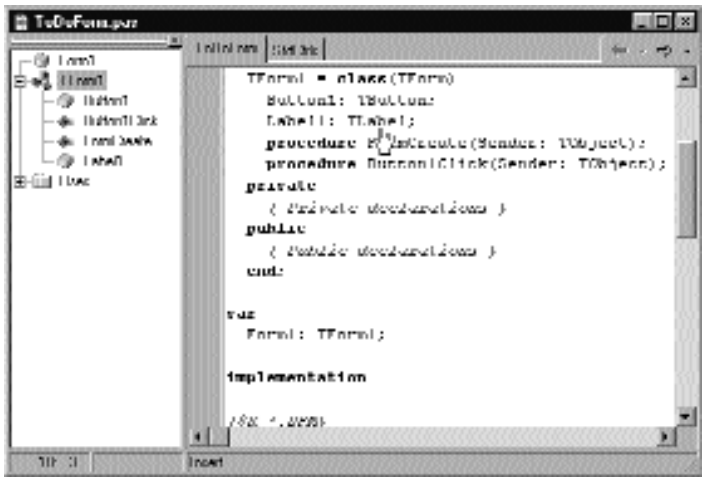
Pretraživanje u editoru

Još jedna funkcija editora AppBrowser je *Tooltip Symbol Insight*. Pomerite pokazivač miša iznad simbola u editoru, a Tooltip će prikazati gde je identifikator deklarisan. Ova funkcija može da bude izuzetno značajna za praćenje identifikatora, klasa i funkcija u okviru aplikacije koju pišete, kao i referenca za izvorni kod Visual Component Library (VCL).

UPOZORENJE

Mada na prvi pogled može izgledati kao dobra ideja, ne možete koristiti Tooltip Symbol Insight da biste saznali koja jedinica deklariše identifikator koji želite da upotrebite. Ukoliko odgovarajuća jedinica nije već uključena, Tooltip se neće pojaviti. ■

Pravi bonus ove fukcije je ipak to što je možete pretvoriti u pomoć pri navigaciji. Kada držite pritisnut taster Ctrl i pomerite pokazivač miša iznad identifikatora, Delphi kreira aktivni link ka definiciji umesto da prikaže Tooltip. Ovi linkovi su prikazani plavom bojom i podvučeni su, što je tipično za web pretraživače, a pokazivač menja oblik u ruku kadgod se nađe iznad linka, kao što možete videti na slici 1.7.



SLIKA 1.7 *Mogućnosti pretraživanja Delphija se aktiviraju kada držite pritisnut taster Ctrl i pomerite pokazivač miša iznad identifikatora*

Na primer, možete pritisnuti taster Ctrl i kliknuti identifikator `TLabel` da biste otvorili definiciju u izvornom kodu VCL. Kako selektujete reference, editor pamti različite pozicije na koje ste skočili, te se možete kretati unapred i unazad — ponovo kao u web pretraživaču. Takođe, možete kliknuti na strelice nadole pored kontrola Back i Forward da biste prikazali detaljan spisak linija izvornog koda na koje ste već skočili, i da biste imali više kontrole nad kretanjem unapred i unazad.

Kako možete skočiti direktno u izvorni kod VCL ako nije deo Vašeg projekta? AppBrowser može pronaći ne samo jedinice iz putanje Search (koje se kompajliraju kao deo Vašeg projekta), već i one koje se nalaze u putanjama Delphi Debug Source, Browsing i Library. Ovi direktorijumi se pretražuju po redosledu koji sam naveo, a možete ih odrediti na strani Directories/Conditionals okvira za dijalog Project Options i na strani Library okvira za dijalog Environment Options. Po definiciji, Delphi dodaje direktorijume izvornog koda VCL u Browsing putanju okruženja, koja ima sledeće deklaracije:

```
$(DELPHI)\source\vc1;$(DELPHI)\source\rt1\Corba
$(DELPHI)\source\rt1\Sys;$(DELPHI)\source\rt1\Win
$(DELPHI)\source\Internet.
```

U ovom nizu putanja, deklaracija `$(DELPHI)` označava direktorijum u kojem je Delphi instaliran.

Class Completion

Treća važna funkcija Delphijevog AppBrowsera je *Class Completion*, koja se aktivira upotrebom kombinacije tastera Ctrl+Shift+C. Dodavanje manipulisanja događajem aplikaciji je brza operacija, jer Delphi automatski dodaje deklaraciju novog metoda za rukovanje događajem u klasu i obezbeđuje Vam strukturu metoda u implementacionom delu jedinice. Ovo je deo Delphijeve podrške vizuelnom programiranju.

Novije verzije Delphija pojednostavljuju život programerima koji dodaju kod uz rukovanje metodima. Nove funkcije generisanja koda se, zapravo, odnose na opšte metode, metode rukovanja porukama i svojstva. Na primer, ukoliko unesete sledeći kod u deklaraciju klase:

```
public
  procedure Hello (MessageText: string);
```

i pritisnete kombinaciju tastera Ctrl+Shift+C, Delphi će Vam obezbediti definiciju metoda u implementacionom delu jedinice, generišući sledeće linije koda:

```
{ Tform1 }
procedure Tform1. Hello(MessageText: string);
begin
end;.
```

Ovo je zaista korisno u poređenju sa tradicionalnim pristupom mnogih programera Delphija koji kopiraju jednu ili više deklaracija, dodaju nazive klasa i na kraju dupliraju kod `begin … end` za svaki kopirani metod.

Class Completion može da funkcioniše i obrnuto. Možete napisati implementaciju metoda direktno njegovim kodom, a zatim pritisnuti kombinaciju tastera Ctrl+Shift+C da biste generisali stavku u deklaraciji klase.

Ukoliko se nakratko vratite na podešavanja Explorera koja su prikazana na slici 1.5, primetićete opciju za Class Completion — možete je upotrebiti da biste kompletirali definiciju svojstva. Ukoliko jednostavno unesete potpuno novu klasu,

```
property X: Integer;
```

i aktivirate Class Completion, Delphi generiše metod `SetX` za svojstvo i dodaje klasi polje `FX`. Rezultujući kod će izgledati ovako:

```
type
  Tform1 = class(TForm)
  private
    FX: Integer;
    procedure SetX(const Value: Integer);
public
  property X: Integer read FX write SetX;
end;

implementation

procedure Tform1.SetX(const Value: Integer);
begin
  FX := Value;
end;.
```

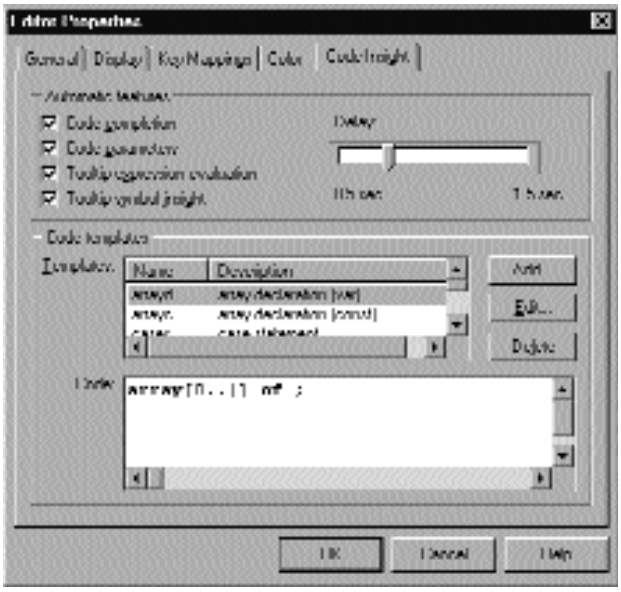
Ovim se zaista spasavate unosa. Zapravo, možete čak delimično kontrolisati kako Class Completion generiše metode svojstva Set i Get, kao što ćete videti u Poglavlju 3 u odeljku posvećenom svojstvima.

Code Insight

Pored Code Explorera, Class Completiona i funkcija za kretanje, Delphi editor još uvek podržava tehnologiju *Code Insight* koja je prvobitno predstavljena u Delphiju 3. Sve u svemu, tehnike Code Insight su zasnovane na stalnoj analizi sintakse u pozadini, kako izvornog koda koji pišete, tako i izvornog koda sistemskih jedinica na koji se refereiše Vaš izvorni kod. Code Insight ima pet mogućnosti.

- **CODE COMPLETION** omogućava da odaberete svojstvo ili metod objekta tako što ćete jednostavno potražiti u spisku ili unošenjem početnih slova. Da biste ga aktivirali, jednostavno unesite naziv objekta, recimo `Button1`, zatim unesite tačku i sačekajte. Da biste primorali program da prikaže listu, pritisnite kombinaciju tastera Ctrl+razmak; da biste uklonili prikaz kada ga ne želite, pritisnite taster Esc. Code Completion Vam, takođe, omogućava da pogledate odgovarajuću vrednost u iskazu dodele. Kada unesete `:=` iza promenljive ili svojstva, Delphi će prikazati spisak ostalih promenljivih ili objekata istog tipa, kao i objekte koji sadrže svojstva tog tipa. Dok je spisak prikazan, možete ga kliknuti desnim tasterom miša da biste promenili redosled elemenata, sortirajući ih prema oblasti delovanja ili prema nazivu.
- **CODE TEMPLATES** Vam omogućava da umetnete jedan od unapred definisanih šablona koda, kao što su recimo složeni iskazi u kojima postoji unutrašnji blok `begin…end`. Code Templates se mora ručno aktivirati tako što ćete upotrebiti kombinaciju tastera Ctrl+J da bi se prikazao spisak svih šablona. Ukoliko unesete nekoliko slova (recimo ključnu reč) pre nego što pritisnete kombinaciju tastera Ctrl+J, Delphi će prikazati spisak samo onih šablona koji počinju tim slovima. Pogledajte dodatak "Prilagođavanje Code Templatesa" da biste videli kako da dodate novi šablon šablonima koje obezbeđuje Delphi.
- **CODE PARAMETERS** prikazuje, u prozoru Tooltip ili prozoru za savet, tip podataka parametra funkcije ili metoda prilikom unosa. Jednostavno unesite naziv funkcije ili metoda i otvorite (levu) zagradu i odmah će se prikazati nazivi parametara i njihovi tipovi. Da biste primorali program da prikaže Code Parameters, možete upotrebiti kombinaciju tastera Ctrl+Shift+razmak. Kao dodatna pomoć, trenutni parametar je prikazan masnim slovima.
- **TOOLTIP EXPRESSION EVALUATION** je funkcija koja je aktivna prilikom otklanjanja grešaka. Prikazuje Vam vrednost identifikatora, svojstva ili izraza koji se nalaze ispod pokazivača miša.
- **TOOLTIP SYMBOL INSIGHT** Vam omogućava da pogledate definiciju identifikatora u oblačiću, što smo ranije pomenuli u odeljku "Pretraživanje u editoru".

Možete uključiti ili isključiti, ili konfigurisati svaku od ovih funkcija na strani Code Insight okvira za dijalog Editor Options koji je prikazan na slici 1.8.



SLIKA 1.8 *Strana Code Insight okvira za dijalog Editor Options, koja Vam omogućava da aktivirate ili isključite svaku od ovih tehnologija i da odredite vreme pauze*

SAVET

Kada kod koji ste napisali nije korektan, Code Insight neće funkcionisati i možda ćete videti samo opštu poruku o grešci kojom je identifikovana takva situacija. Moguće je prikazati specifične greške Code Insight u panelu Message (koji mora biti već otvoren; ne otvara se automatski da bi prikazao greške prilikom kompajliranja). Da biste aktivirali ovu funkciju, potrebno je da podesite još jednu nedokumentovanu stavku u Registryju, podešavajući vrednost string ključa `Delphi\5.0\Compiling\ShowCodeInsiteErrors` u vrednost `"1"`. ■

PRILAGOĐAVANJE CODE TEMPLATESA

Možete upotrebiti Code Templates da biste dodelili naziv često korišćenom izrazu ili pozivu funkcije. Na primer, ukoliko često koristite funkciju `MessageDlg`, možda želite da dodate šablon za tu funkciju. Pređite na stranu Code Insight okvira za dijalog Evironment Options, kliknite kontrolu Add u oblasti Code Templates, unesite naziv novog šablona (na primer, **mess**), unesite opis, a zatim dodajte naredni tekst telu šablona u kontroli Code memo:

```
MessageDlg ('|',
  mtInformation, [mbOK], 0);.
```

Sada, svaki put kada imate potrebu da kreirate dijalog za poruku, jednostavno unesite **mess** i pritisnite kombinaciju tastera Ctrl+J i dobićete ceo tekst. Vertikalna linija (pipe) označava poziciju u okviru izvornog koda gde će se nalaziti kursor posle dodavanja šablona. Potrebno je da odaberete poziciju tako da to bude pozicija na kojoj ćete početi unos, da biste kompletirali kod koji generiše šablon (u ovom slučaju da unesete tekst koji će se prikazati u poruci).

Code Templates nema direktnu vezu sa ključnim rečima jezika; šabloni su više opšti mehanizam. Code Templates se čuva u fajlu `DELPHI32.DCI` i trebalo bi da je moguće da kopirate ovaj fajl i da Vaše šablone prenesete na različite mašine. Spojiti dva fajla sa šablonima nije lako, i zbog toga još uvek ne postoje alati kojima je moguće dodati više šablona mašini.

Još tastaturnih prečica editora

U editoru postoji još mnogo tastaturnih prečica koje zavise od stila editora koji ste odabrali. Evo nekih manje poznatih tastaturnih prečica, od kojih je većina korisna:

- Ctrl+Shift i neki taster sa brojem između 0 i 9 aktivira oznaku (bookmark) koja je naznačena na margini na strani editora. Da biste se vratili na oznaku, možete pritisnuti Ctrl i taster na kojem je broj. Upotrebljivost oznaka u editoru je ograničena činjenicom da nova oznaka može prepisati oznaku koja nije stalna; oznake se gube kada zatvorite fajl.
- Ctrl+E aktivira pretraživanje uvećanjem. Možete upotrebiti kombinaciju tastera Ctrl+E i uneti reč koju želite da pronađete, a da ne morate da koristite specijalni okvir za dijalog i kliknete Enter da biste obavili pretraživanje.
- Ctrl+Shift+I uvlači više linija koda odjednom. Broj razmaka koji se koristi, je onaj koji je određen opcijom Block Indent na strani Editor okvira za dijalog Environment. Ctrl+Shift+U je odgovarajuća kombinacija kojom se poništava uvlačenje koda.
- Ctrl+O+U menja velika u mala slova koda i obrnuto; možete, takođe, upotrebiti kombinaciju tastera Ctrl+K+E da biste velika slova promenili u mala, a kombinaciju Ctrl+K+F da biste mala slova pretvorili u velika.
- Ctrl+Shift+R započinje snimanje makroa, koji kasnije možete upotrebiti ukoliko pritisnete kombinaciju tastera Ctrl+Shift+P. Makro snima sve što otkucate, pomeranja i uklanjanja u fajlu sa izvornim kodom. Upotreba makroa jednostavno ponavlja sekvencu — operacija koja nema nekog značaja kada pređete u drugi fajl izvornog koda. Ja tek treba da pronađem upotrebnu vrednost ove tehnike, mada pretpostavljam da je Borland koristi za testiranje.
- Dok držite pritisnut taster Alt, mišem možete označiti četvorougaone oblasti editora, ne samo uzastopne linije i reči.

Form Designer

Još jedan prozor Delphija, u kojem ćete često raditi, je Form Designer, vizuelni alat koji Vam pomaže da smestite komponente na formular. U Form Designeru možete odabrati komponentu mišem ili pomoću Object Inspectora; zgodna funkcija kada se kontrola nalazi ispod neke druge kontrole ili je kontrola mala. Ukoliko jedna kontrola u potpunosti prekriva drugu kontrolu, možete pritisnuti taster Esc da biste selektovali roditeljsku kontrolu kontrole koja je selektovana. Taster Esc možete pritisnuti jednom ili više puta da biste selektovali formular, ili pritisnuti i držati pritisnut taster Shift kada kliknete selektovanu komponentu. Na ovaj način ćete iz selekcije ukloniti komponentu i po definiciji selektujete formular.

SAVET

Šta ako je potrebno pomeriti kontrolu prevlačenjem prilikom dizajniranja, ali se u oblasti nalazi kontrola? Jednostavno prevucite kontrolu i pritisnite taster Esc (dok držite pritisnut taster miša) da biste se prebacili na prevlačenje roditeljske kontrole. ■

Postoje dve mogućnosti pri upotrebi miša za određivanje pozicije komponente. Možete odrediti vrednosti za svojstva `Left` i `Top`, ili možete upotrebiti kursor-tastere dok držite pritisnut taster Ctrl. Upotreba kursor-tastera je naročito korisna za fino pozicioniranje elemenata. (Opcija Snap to Grid funkcioniše samo kada koristite miša.) Slično tome, kada koristite kursor-tastere dok držite pritisnut taster Shift, možete fino podesiti veličinu komponente. (Ukoliko pritisnete kombinaciju tastera Ctrl+Shift i neki od kursor-tastera, komponenta će se pomeriti za veličinu mreže.) Na nesreću, tokom ovih finih promena pomoć za poziciju i veličinu komponente se ne prikazuje.

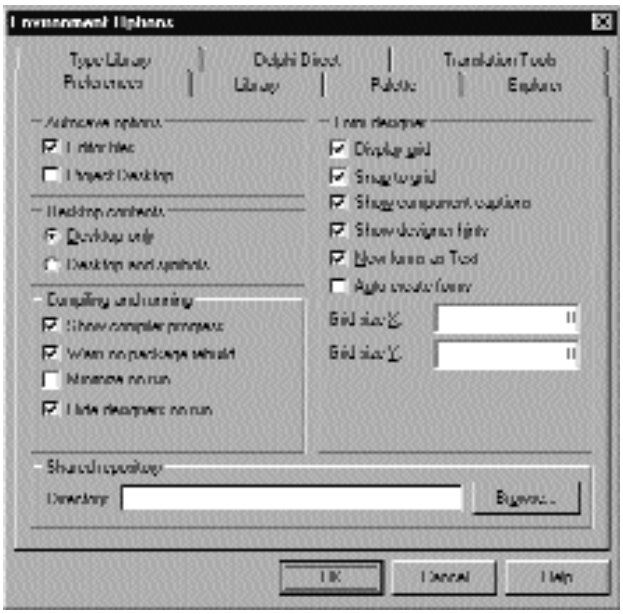
Da biste poravnali više komponenata, ili da biste im dodelili jednaku veličinu, možete selektovati nekoliko komponenata i podesiti svojstva `Top, Left, Width` ili `Height` za sve komponente odjednom. Da biste selektovali nekoliko komponenata, možete kliknuti komponente mišem dok držite pritisnut taster Shift ili, ukoliko sve komponente mogu stati u četvorougaonu oblast, možete prevući mišem da biste "nacrtali" četvorougao koji ih obuhvata. Kada ste selektovali više komponenata, možete odrediti njihove relativne pozicije upotrebivši okvir za dijalog Alignment (upotrebom komande Align iz kontekst menija formulara) ili paletu Alignment (kojoj možete pristupiti preko komande menija View➥Alignment Pallete).

Kada završite dizajniranje formulara, možete upotrebiti komandu Lock Controls iz menija Edit da biste izbegli slučajnu promenu pozicije komponente na formularu. Ovo je naročito korisno jer zapravo ne postoji prava operacija Undo za formulare (samo Undelete), ali vrednosti nisu nepromenljive.

Među ostalim funkcijama, Form Designer nudi brojnu pomoć Tooltip.

- Kada pomerite pokazivač iznad komponente, savet će prikazati naziv i tip komponente. Ovo je alternativa podešavanju okruženja Show Component Captions, za koje se ja uvek trudim da je aktivno.
- Kada promenite veličinu kontrole, pomoć prikazuje aktuelnu veličinu (svojstva `Width` i `Height`). Naravno, ova funkcija postoji samo za kontrole, ne za nevizuelne komponente (koje su u Form Designeru naznačene ikonama).
- Kada pomerite komponentu, pomoć prikazuje aktuelnu poziciju (svojstva `Left` i `Top`).

Na kraju, ono što bi mogla da bude najvažnija nova funkcija Form Designera Delphija 5, je da možete sačuvati DFM (Delphi Form Module) fajlove kao tekstualne fajlove umesto tradicionalnog binarnog formata. Ovu opciju možete uključiti ili isključiti za svaki formular ponaosob upotrebom kontekst menija Form Designera, ili možete odrediti vrednost za sve novokreirane formulare na strani Preferences okvira za dijalog Environment Options (videti sliku 1.9).



SLIKA 1.9 *Strana Preferences okvira za dijalog Environment Options u Delphiju 5 Vam omogućava da odredite da li će se formulari kreirati po definiciji, i da li će DFM fajlovi sadržati tekst*

Na istoj strani možete, takođe, odrediti da li će se sekundarni formulari programa automatski kreirati prilikom pokretanja; odluku uvek možete poništiti za pojedine formulare (koristeći stranu Forms okvira za dijalog Project Options). Međutim, najočiglednija razlika između Delphija 5 i prethodnih verzija, prilikom rada sa formularaima, je Object Inspector.

Dobrodošla pomoć je mogućnost čuvanja DFM fajlova kao tekst falova; omogućava Vam bolje operisanje sistemima kontrole verzija. Programeri neće dobiti stvarnu prednost ovom funkcijom, jer ste i ranije mogli da otvorite binarni DFM fajl u Delphi editoru upotrebivši kontekst meni dizajnera. Sistemima kontrole vezija, s druge strane, je potrebno da sačuvaju tekstualnu verziju DFM fajlova da bi mogli da ih uporede i pronađu razlike između dve verzije istog fajla. Ovo je verovatno predstavljeno u Delphiju 5 uz Team-Source interfejs sistema kontrole verzije, koji ćemo razmatrati u Poglavlju 19.

U svakom slučaju, zapamtite da ukoliko koristite DFM fajlove kao tekst, Delphi će ih ipak konvertovati u binarni format resursa pre nego što ih uključi u izvršni fajl Vašeg programa. DFM se povezuju u izvršne programe u binarnom formatu da bi se smanjila veličina izvršnog fajla (mada se zapravo ne kompresuju) i da bi se poboljšale performanse prilikom izvršavanja (mogu se brže učitati).

NAPOMENA

Ranije verzije Delphi IDE-a neće prepoznati tekstualne DFM fajlove. Kada otvorite tekstualni DFM fajl u Delphiju 4 (ili nekoj ranijoj verziji), dobićete poruku o grešci. Da biste to izbegli, morate prvo upotrebiti Delphi 5 da biste konvertovali DFM fajl u binarni format, koristeći kontekst meni Form Designera. (Na kompjuteru na kojem nema Delphija 5, možete upotrebiti alat komandne linije `CONVERT` Delphija 4.) Kada otvorite postojeći DFM u Delphi 5 IDE-u, prvobitni DFM format će biti sačuvan (izuzev ukoliko ga eksplicitno ne promenite upotrebom Text DFM elementa kontekst menija), čime imate mogućnost da ponovo otvorite isti formular u prethodnoj verziji Delphija. ■

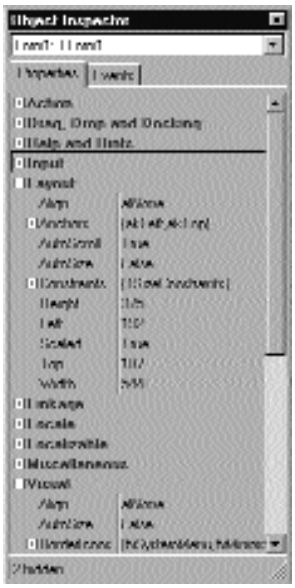
Object Inspector u Delphiju 5

Ukoliko ste ranije koristili Delphi, odmah ćete primetiti neke novine u Object Inspectoru. Najvažnije izmene se tiču grafičke liste i kategorija svojstava.

Prvi element je najlakše koristiti. Lista za svojstvo u Object Inspectoru može da sadrži grafičke elemente. Mnoga relevantna svojstva koriste ovu funkciju po definiciji: `Color`, `Cursor` i njegove varijacije, `ImageIndex` — opšte svojstvo komponenata povezanih ImageListom (kao što su, recimo, akcija, element menija ili kontrola palete alata), stilovi Pen i Brush i nekoliko drugih. Na primer, slika 1.10 prikazuje spisak kursora (svojstva `Cursor`). Naravno, programeri Delphi komponenata i dodataka će moći da prilagode ovu funkciju, pa ćete u budućnosti videti mnogo više elemenata u listi. Pogledajte dodatak "Lista fontova u Object Inspectoru" da biste videli jednostavno prilagođavanje ovog prozora.

SLIKA 1.10 *Grafička lista Delphi 5 Object Inspectora koja prikazuje kursore koji su na raspolaganju*

Potrebno je malo više vremena da biste se upoznali sa kategorijama svojstava. Da biste razumeli ovu funkciju, potrebno je prvo da je učinite vidljivom. Da biste svojstva prikazali po kategorijama umesto po nazivu, kliknite desnim tasterom miša Object Inspector i odaberite odgovarajuću opciju Arrange iz kontekst menija. Efekat izbora koji ste načinili možete videti na slici 1.11.



SLIKA 1.11 *Efekat uređenja svojstava prema kategoriji*

Ukoliko pažljivo pogledate sliku, primetićete nešto čudno — svojstvo `Align` je dostupno u dvema različitim kategorijama. Ovo je opšte pravilo; kategorije nisu ekskluzivne i svojstvo može da bude registrovano za više različitih kategorija.

SLIKA 1.12 *Možete sakriti svojstva nekih kategorija iako su uređena po nazivima*

Kategorije imaju prednost smanjenja složenosti Object Inspectora. Možete upotrebiti podmeni View iz kontekst menija da sakrijete svojstva datih kategorija, bez obzira na način na koji su prikazana (dakle, iako više volite tradicionalni prikaz i uređenje po nazivima, još uvek možete da sakrijete svojstva nekih kategorija). Na primer, na slici 1.12 možete videti svojstva formulara uređena po nazivu, ali samo svojstva iz kategorija Visual i Input. Zapravo, kao što možete videti na statusnoj liniji Object Inspectora, sakrivena su 44 svojstva. Uređenje i vidljivost koje odaberete, će, takođe, uticati i na događaje.

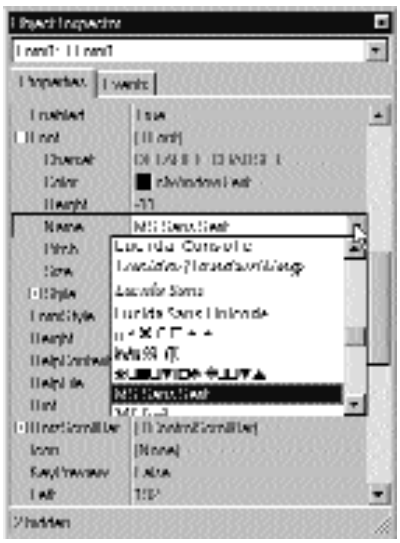
SAVET

Još jedna nova funkcija Delphi 5 Object Inspectora je mogućnost selektovanja komponente na koju se referiše svojstvom. Da biste to učinili, levim tasterom miša dva puta kliknite vrednost svojstva dok držite pritisnut taster Ctrl. Na primer, ukoliko imate komponentu MainMenu na formularu i posmatrate svojstva formulara u Object Inspectoru, možete selektovati komponentu MainMenu tako što ćete se pomeriti do svojstva MainMenu na formularu i pritisnuti taster Ctrl i dva puta kliknuti vrednost svojstva. Ovim ćete selektovati glavni meni naznačen kao vrednost svojstva u Object Inspectoru. Ova funkcija može da bude naročito korisna kada imate povezane komponente; na primer, kada koristite više izvora podataka i komponenata skupova podataka. ■

LISTA FONTOVA U OBJECT INSPECTORU

Delphi 5 Object Inspector sadrži grafičku listu za nekoliko svojstava. Možda želite da dodate onu koja prikazuje aktuelnu sliku fonta koji selektujete, za podsvojstvo `Name` svojstva `Font`. Ova mogućnost je, zapravo, ugrađena u Delphi 5, ali je isključena jer je na većini kompjutera instaliran veliki broj fontova i njihovo renderovanje može umnogome usporiti Vaš kompjuter. Ukoliko želite da uključite ovu funkciju, potrebno je da instalirate paket Delphi koji omogućava globalnu promenljivu `FontNamePropertyDisplayFontNames` jedinice Dsgnlitf. Ja sam to učinio u paketu OiFont Pk, koji možete pronaći među primerima programa ovog poglavlja.

Kada je paket instaliran, možete preći na svojstvo `Font` bilo koje komponente i upotrebiti grafički meni Name, kao što je ovde prikazano.



Postoji i drugo, složenije prilagođavanje Object Inspectora koje se meni dopada i koje često koristim — font za ceo Object Inspector, da bi tekst bio čitljiviji. Ova funkcija je naročito korisna za javne prezentacije. Paket za instalaciju fontova možete pronaći u Object Inspectoru na mom web sajtu, `www.marcocantu.com`.

Tajne palete Component

Component Palette je veoma lako koristiti, ali postoji nekoliko stvari koje možda ne poznajete. Postoje četiri jednostavna načina za postavljanje komponente na formular.

- Posle selektovanja kontrole na paleti kliknite formular da biste odredili poziciju kontrole i kliknite i prevucite mišem da biste joj odredili veličinu.
- Posle selektovanja bilo koje komponente jednostavno kliknite formular da biste je postavili sa unapred određenom visinom i dužinom.
- Dva puta kliknite ikonu na paleti da biste dodali komponentu tog tipa na sredinu formulara.
- Pritisnite taster Shift i kliknite ikonu komponente da biste postavili nekoliko komponenata istog tipa na formular. Da biste prekinuli ovu operaciju, jednostavno kliknite standardni selektor (ikonu sa strelicom) na levoj strani Component Palette.

Možete odabrati komandu Properties iz kontekst menija palete da biste u potpunosti promenili uređenje komponenata na raznim stranama palete, sa mogućnošću dodavanja novih elemenata ili mogućnošću premeštanja elemenata sa jedne strane na drugu. Na rezultujućoj strani Properties možete jednostavno prevući komponentu iz liste Components u listu Pages da biste pomerili komponentu sa jedne strane na drugu.

SAVET

Kada imate previše strana u paleti Component, potrebno je da upotrebite klizače da biste došli do komponente. Postoji jednostavan trik koji možete upotrebiti u ovom slučaju: promenite nazive strana dajući im kraće nazive tako da sve strane stanu na ekran. Očigledno — jednom ste o tome razmišljali. ■

Stvarno nedokumentovana funkcija Component Palette je aktiviranje "hot-track". Određivanjem specijalnih tastera u Registryju možete jednostavno selektovati stranu palete prelaskom na jezičak, a da ne morate da kliknete mišem. Ista funkcija se može dodeliti klizačima komponenata na obe strane palete, koji se prikazuju kada strana sadrži previše komponenata.

Da biste aktivirali ovu skrivenu funkciju, potrebno je da dodate ključ Extras pod `HKEY_CURRENT_USER\Software\Borland\Delphi\5.0`. Pod ovim ključem potrebno je da unesete dve string vrednosti, `AutoPaletteSelect` i `AutoPaletteScroll`, i da svakoj dodelite vrednost stringa `"1"`.

Definisanje rukovanja događajima

Postoji nekoliko tehnika koje možete upotrebiti da biste definisali rukovanje događajem komponente:

- selektujte komponentu, pređite na stranu Events i dva puta kliknite na belu površinu na desnoj strani događaja, ili unesite naziv na tu površinu i pritisnite taster Enter;
- za mnoge kontrole možete dva puta kliknuti kontrolu da biste izvršili unapred određenu akciju, kojom se dodaje rukovanje događajima `OnClick`, `OnChange` ili `OnCreate`.

Kada želite da uklonite rukovanje događajem koji ste napisali iz izvornog koda Delphi aplikacije, možete ukloniti sve reference. Ipak, bolji način je da uklonite sav kod iz odgovarajuće procedure, ostavljući samo deklaraciju i ključne reči begin i end. Tekst bi trebalo da bude isti kao da ga je Delphi automatski generisao kada ste prvi put odlučili da upravljate događajem. Kada sačuvate ili kompajlirate projekat, Delphi uklanja prazne metode iz izvornog koda i iz opisa (uključujući i reference sa strane Events Object Inspectora). Suprotno, da biste zadržali rukovanje događajem koje je još uvek prazno, razmislite o dodavanju komentara (čak jednostavno dodajte karaktere //), i onda neće biti uklonjen.

Kopiranje i smeštanje komponenata

Interesantna funkcija Form Designera je mogućnost kopiranja i smeštanja komponenata sa jednog formulara na drugi formular ili dupliranje komponenata formulara. Tokom ove operacije Delphi duplira sva svojstva i zadržava sva povezana rukovanja događajima i, ukoliko je potrebno, menja naziv kontrole (jer naziv mora biti jedinstven u okviru formulara).

Takođe je moguće kopirati komponente iz Form Designera u editor i obrnuto. Kada kopirate komponentu na Clipboard, Delphi, takođe, smešta i tekstualni opis. Možete čak i da promenite tekst verzije komponente, kopirati tekst na Clipboard, a zatim ga smestiti natrag u formular kao novu komponentu. Na primer, ukoliko na formular smestite kontrolu, kopirate je, a zatim smestite u editor (koji može da bude Delphijev editor izvornog koda ili bilo koji tekst procesor), dobićete sledeći opis:

```
object Button1: TButton
  Left = 152
  Top = 104
  Width = 75
  Height = 25
  Caption = 'Button1'.
  Tab0rder = 0
end
```

Sada, ukoliko promenite naziv objekta, njegov naslov ili poziciju, na primer, ili dodate novo svojstvo, ove promene se mogu kopirati i smestiti natrag na formular. Evo primera nekoliko izmena:

```
object Button1: TButton
  Left = 152
  Top = 104
  Width = 75
  Height = 25
  Caption = 'My Button'
  Taborder = 0
  Font.Name = 'Arial'
end
```

Kopiranje ovog opisa i njegovo smeštanje na formular će kreirati kontrolu na naznačenoj poziciji, a naslov kontrole će biti *My Button* u fontu Arial.

Da biste iskoristili ovu tehniku, potrebno je da znate kako da izmenite tekstualnu reprezentaciju komponente, koja svojstva su valjana za određenu komponentu i kako da unesete vrednosti za tekstualna svojstva, podesite svojstva i druga specijalna svojstva. Kada Delphi interpretira tekstualni opis komponente ili formulara, može promeniti vrednosti drugih svojstava koja se odnose na svojstva koja ste promenili, a može i promeniti poziciju komponenata tako da ne preklapa prethodnu kopiju. Naravno, ukoliko napišete nešto što je potpuno pogrešno i pokušate da to smestite na formular, Delphi će prikazati poruku o grešci obaveštavajući Vas o tome šta je pogrešno.

Možete selektovati nekoliko komponenata i sve ih kopirati odjednom, bilo na drugi formular bilo u editor teksta. To može da bude korisno onda kada je potrebno raditi na nizu sličnih komponenata. Možete jednu kopirati u editor, replicirati je više puta, načiniti neophodne izmene, a zatim celu grupu smestiti ponovo na formular.

Od šablona komponenata do okvira

Kada kopirate jednu ili više komponenata sa jednog formulara na drugi, jednostavno kopirate sva njihova svojstva. Mnogo moćniji pristup je da kreirate šablon komponente (component template), čime stvarate kopiju kako svojstava tako i izvornog koda rukovanja događajima. Kada smestite šablon u novi formular, selektovanjem pseudokomponente sa palete, Delphi će replicirati izvorni kod rukovanja događajima na novom formularu. Da biste kreirali šablon komponente, selektujte jednu ili više komponenata i odaberite komandu menija Component➥Create Component Template. Na ovaj način ćete otvoriti okvir za dijalog Component Template Information (videti sliku 1.13) u koji možete uneti naziv šablona, stranu palete Component na kojoj treba da se prikaže i ikonu.

SLIKA 1.13 *Okvir za dijalog Component Template Information*

Po definiciji, naziv šablona je naziv prve komponente koju ste selektovali za kojom sledi reč *Template*. Unapred određena ikona šablona je ikona prve komponente koju ste selektovali, ali je možete promeniti izborom fajla sa ikonom. Naziv koji dodelite šablonu komponente će se korisititi kao opis u paleti Component (kada Delphi prikaže oblačić).

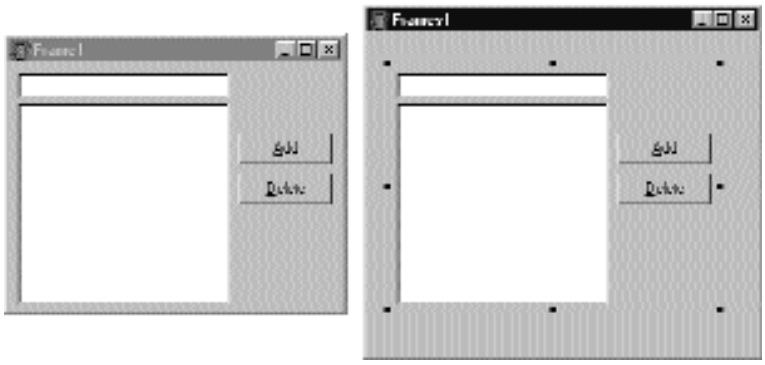
Sve informacije o šablonima komponenata se čuvaju u jednom fajlu, DELPHI23.DCT, ali izgleda nije moguće iz fajla dobiti informacije i izmeniti šablon. Ono što ipak možete učiniti je da postavite šablon komponente na potpuno novi formular, izmenite ga i ponovo instalirate kao šablon komponente *koristeći isti naziv.* Na ovaj način možete zameniti prethodnu definiciju.

SAVET

Grupa programera Delphija može deliti šablone komponenata čuvajući ih u zajedničkom direktorijumu, dodajući u Registry stavku `CCLibDir` pod ključem `Software\Borland\Delphi\5.0\Component Templates`. ■

Šabloni komponenata su zgodni kada je na različitim formularima potrebna ista grupa komponenata i odgovarajuće rukovanje događajima. Problem nastaje kada jednom postavite instancu šablona na formular; Delphi stvara kopiju komponenata i njihovog koda koji nije više u vezi sa šablonom. Ne postoji način da izmenite samu definiciju šablona, i sasvim sigurno nije moguće da se izmene odslikaju na sve formulare koji koriste šablon. Da li ja tražim previše? Nikako. To je ono što možete postići novom tehnologijom okvira (frames) u Delphiju 5.

Okvir je vrsta panela sa kojim možete da radite prilikom dizajniranja na način sličan radu sa formularom. Jednostavno kreirate novi okvir, na njega postavite nekoliko kontrola i dodate kod rukovanju događajima. Kada je okvir spreman, možete otvoriti formular, odabrati pseudokomponentu Frame sa strane Standard Component Palette i odabrati jedan od mogućih okvira (aktuelnog projekta). Kada postavite okvir na formular, biće prikazan kao da su komponente kopirane. Ukoliko izmenite prvobitni okvir (u njegovom dizajneru), izmene će se odslikati na svaku instancu na formularu. Možete pogledati jednostavan primer, nazvan Frames1, na slici 1.14. Snimak ekrana zapravo ne znači mnogo; trebalo bi da otvorite program ili ponovo izradite sličan, ukoliko želite da se poigrate okvirima.



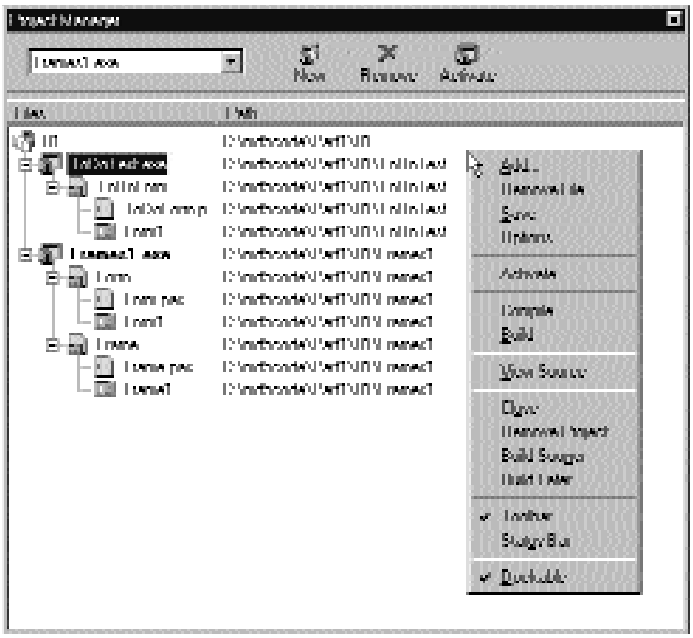
SLIKA 1.14 *Primer Frames1 demonstrira upotrebu okvira. Okvir (na levoj strani slike) i njegove instance na formularu (na desnoj strani slike) su sinhronizovani*

Slično formularima, okviri definišu klase, tako da se mnogo lakše uklapaju u VCL objektni model nego Component Templates. Poglavlje 4 nudi detaljno objašnjenje VCL-a, a sadrži i detaljniji opis okvira. Kao što možete pretpostaviti iz ovog kratkog uvoda, okviri su moćna nova tehnika.

Upravljanje projektima

Jedna od novih funkcija Delphi 4 IDE-a bila je višeciljni Project Manager (View➥Project Manager). Project Manager radi sa *grupom* projekta koja može imati jedan ili više projekata u sebi. Na primer, grupa projekta može sadržati DLL i izvršni fajl, ili više izvršnih fajlova.

Na slici 1.15 možete videti Project Manager sa grupom projekta primera ovog poglavlja. Kao što se vidi, Project Manager je zasnovan na prikazu drveta, kojim se prikazuje hijerarhijska struktura grupe projekta, projekata i svih formulara i jedinica koje sačinjavaju svaki od projekata. Možete koristiti jednostavnu paletu alata i složenije kontekst menije Project Managera da biste obavljali operacije. Kontekst meniji su kontekst senzitivni; opcije kontekst menija zavise od selektovanog elementa. Postoje elementi menija za dodavanje novih ili postojećih projekata grupi projekta, za kompajliranje ili izradu određenog projekta, ili za otvaranje jedinice.



SLIKA 1.15 *Delphijev višeciljni Project Manager*

Od svih projekata u grupi samo je jedan aktivan, i to je projekat sa kojim operišete kada odaberete komandu kao što je Project➥Compile. Podmeni Project glavnog menija sadrži dve komande koje možete upotrebiti za kompajliranje ili izradu svih projekata grupe. (Prilično je neobično što ove komande nisu dostupne u kontekst meniju Project Manager grupe projekta.)

Kada je potrebno da izradite više projekata, možete odrediti relativni redosled upotrebom komandi Build Sooner i Build Later. Ove dve komande u osnovi iznova uređuju projekte u listi.

Delphi 5 dodaje nove funkcije Project Manageru. Sada možete prevući fajlove sa izvornim kodom iz Windowsovog direktorijuma ili Windows Explorera u projekat u prozoru Project Managera da biste ih dodali projektu. Na nesreću, ne možete prevući postojeći projekat ili fajl paketa da biste ga dodali celoj grupi projekta. Takođe, možete prevlačiti iz jednog projekta u drugi projekat iste grupe projekta.

Još jedna velika prednost je da Project Manager automatski kao aktuelni projekat određuje onaj projekat na kome trenutno radite, na primer, otvarajući fajl. Lako možete videti koji projekat je selektovan, i promeniti ga koristeći polje na vrhu formulara.

SAVET

Pored dodavanja Pascal fajlova i projekata, možete dodati Windows resurs fajlove Project Manageru; ovi fajlovi se kompajliraju uz projekat. Jednostavno pređite na projekat, odaberite Add iz kontekst menija i odaberite Resource File (*.rc) za tip fajla. Ovaj resurs će se automatski vezati za projekat, čak i bez odgovarajuće direktive `$R`. ■

Delphi čuva grupu projekta pod novom `.BPG` ekstenzijom, što je skraćenica za Borland Project Group. Ova funkcija dolazi iz C++ Buildera i ranijih kompajlera Borland C++, istorije koja je jasno vidljiva kada otvorite izvorni kod grupe projekta, što je u osnovi makefile C/C++ razvojnog okruženja. Evo jednostavnog primera.

```
#------------------------------------------------------------------------------
VERSION = BWS.01
#------------------------------------------------------------------------------
! ifndef ROOT
ROOT = $(MAKEDIR)\..
! endif
#------------------------------------------------------------------------------
MAKE = $(ROOT)\bin\make.exe -$(MAKEPLAGS) -f$**
DCC = $(ROOT)\bin\dcc32.exe $**
BRCC = $(ROOT)\bin\brcc32.exe $**
#------------------------------------------------------------------------------
PROJECTS = Projectl.exe
#------------------------------------------------------------------------------
default: $(PROJECTS)
#------------------------------------------------------------------------------

Projectl.exe:  Projectl.dpr
       $(DCC)
```

Opcije projekta

Project Manager ne omogućava da odredite opcije dva različita projekta odjednom. Ono što umesto toga možete učiniti, je da pozovete okvir za dijalog Project Options iz Project Managera za svaki projekat. Prva strana Project Options (Forms) prikazuje listu formulara koje bi trebalo automatski kreirati prilikom pokretanja programa i formulare koje kreira sam program. Naredna strana (Application) se koristi za određivanje naziva aplikacije i naziva Help fajla, kao i za izbor

ikona. Ostale opcije okvira za dijalog Project Options se odnose na Delphi kompajler i linker, informacije o verziji i upotrebi paketa prilikom izvršavanja.

Postoje dva načina za određivanje opcija kompajlera. Jedan je upotreba strane Compiler okvira za dijalog Project Options. Drugi je određivanje ili uklanjanje pojedinih opcija u izvornom kodu upotrebom komandi `{$X+}` ili `{$X-}`, gde bi X trebalo zameniti opcijom koju želite da upotrebite. Drugi pristup je mnogo fleksibilniji jer Vam omogućava da promenite opciju za određeni fajl izvornog koda, ili čak za nekoliko linija koda. Opcije na nivou koda preinačuju opcije na nivou kompajliranja.

Sve opcije okvira za dijalog Project Options se automatski čuvaju sa projektom, ali u zasebnom fajlu sa ekstenzijom .DOF. To je tekstualni fajl koji je lako izmeniti. Ne bi trebalo da uklonite ovaj fajl ukoliko ste promenili bilo koju od unapred određenih opcija. Delphi, takođe, čuva opcije kompajlera u drugom CFG formatu fajla, za kompajliranje sa komandne linije.

Još jedna alternativa za čuvanje opcija kompajlera je da pritisnete kombinaciju tastera Ctrl+O+O (pritisnite dva puta taster O dok držite pritisnut taster Ctrl). Na ovaj način umećete direktive kompajlera, koje odgovaraju aktuelnim opcijama projekta, na vrh aktuelne jedinice, kao u sledećem listingu.

```
{$A+,B-,C+,D+,E-,F-,G+,H+,I+,J+,K-,L+,M-,N+,O+,P÷,Q-,
R-,S-,T-,U-,V+,W-, X+,Y+,Z1}

{$MINSTACKSIZE $00004000}

{$MAXSTACKSIZE $00100000}

{$IMAGEBASE $00400000}

{$APPTYPE GUI}
```

Kompajliranje i izrada projekata

Postoji nekoliko načina za kompajliranje projekta. Ukoliko ga pokrenete (pritiskom tastera F9 ili ukoliko kliknete ikonu Run na paleti alata), Delphi će prvo kompajlirati projekat. Kada Delphi kompajlira projekat, kompajlira samo fajlove koji su izmenjeni.

Ukoliko odaberete Compile➥Build All, svaki fajl se kompajlira iako nije izmenjen. Ova druga komanda Vam neće često biti potrebna, jer Delphi obično može da prepozna fajlove koji su izmenjeni i kompajlira ih po potrebi. Jedini izuzetak je kada promenite neke opcije projekta. U tom slučaju potrebno je da upotrebite komandu Build All da bi nove opcije imale efekta.

Da bi izradio projekat, Delphi prvo kompajlira svaki od fajlova sa izvornim kodom, generišući Delphi kompajliranu jedinicu (DCU). (Ovaj korak se izvršava samo ukoliko DCU fajl nije ažuriran.) Drugi korak, koji obavlja linker, je spajanje svih DCU fajlova u izvršni fajl, opciono sa kompajliranim kodom iz biblioteke VCL (ukoliko niste odlučili da koristite pakete u vreme izvršavanja). Treći korak je povezivanje u izvršni fajl bilo kojeg od resurs fajlova, kao što je RES fajl projekta, koji sadrži glavnu ikonu i DFM fajlove formulara. Bolje ćete razumeti korake kompajliranja i lakše ćete pratiti korake ukoliko uključite opciju Show Compiler Progress (na strani Preferences okvira za dijalog Environment Options).

UPOZORENJE

Delphi nije uvek u stanju da pravilno vodi računa kada treba ponovo izraditi jedinice koje su zasnovane na drugim jedinicama koje ste izmenili. To je svakako tačno u slučajevima (a ima ih mnogo) kada korisnikova intervencija poremeti logiku kompjutera. Na primer, promena naziva fajlova, izmena fajlova sa izvornim kodom van IDE-a, kopiranje starijih izvornih fajlova ili DCU fajlova na disk, ili ukoliko imate više kopija jedinice izvornog fajla u putanji za pretraživanje, može dovesti do prekida kompajliranja. Svaki put kada kompajler prikaže neku čudnu grešku, prva stvar koju bi trebalo da učinite je da pokušate izvršavanje komande Build All da biste ponovo sinhronizovali funkciju "make" sa aktuelnim fajlovima na disku. ■

Komanda Compile se može koristiti samo kada ste učitali projekat u editor. Ukoliko nema aktivnih projekata, a Vi učitate Pascal izvorni fajl, ne možete da ga kompajlirate. Ipak, ukoliko učitate izvorni fajl *kao da je projekat*, možete da prevarite kompjuter, i moći ćete da ga kompajlirate. Da biste to učinili, jednostavno odaberite kontrolu Open Project sa palete alata i učitajte PAS fajl. Sada možete proveriti sintaksu ili ga kompajlirati, formirajući DCU.

Ranije sam pomenuo da Delphi omogućava da koristite pakete u vreme izvršavanja, što utiče na distribuiranje programa više nego proces kompajliranja. Delphi paketi su dinamičke biblioteke za povezivanje (DLL) koje sadrže komponente Delphija. Korišćenjem paketa izvršni fajl može da bude daleko manji. Ipak, program se neće pokrenuti ukoliko odgovarajuće dinamičke biblioteke za povezivanje (kao što je paket vcl50.bpl, koji je prilično veliki) nisu dostupne na kompjuteru na kome se program izvršava.

Ukoliko dodate veličinu dinamičke biblioteke veličini malog izvršnog fajla, ukupna količina prostora potebna na disku, za očigledno manji program, izrađen upotrebom paketa, je mnogo veća od količine prostora većeg samostalnog izvršnog fajla. Naravno, ukoliko imate više aplikacija na jednom sistemu, možete sačuvati dosta prostora i memorije prilikom izvršavanja. Upotreba paketa je česta, ali nije uvek preporučljiva. Razmatraću detaljno sve implikacije paketa u Poglavlju 13, u kojem ćemo izraditi neke pakete, i u Poglavlju 14, koje je posvećeno DLL-ovima i paketima.

SAVET

Nećete morati da koristite paket vcl50.bpl ukoliko Vam je potreban samo mali skup VCL jedinica. Možete kreirati Vaš sopstveni mali VCL paket ukoliko ga ne nazovete vcl50.bpl. ■

U oba slučaja Delphi izvršni fajlovi se veoma brzo kompajliraju, a brzina rezultujuće aplikacije se može porediti sa programima C ili C++. Delphi kompajlirani kod se izvršava najmanje pet puta brže od ekvivalentnog koda kod alata koji interpretiraju ili polukompajliraju.

USLOVNA KOMPILACIJA ZA RAZLIČITE VERZIJE DELPHIJA

Možete proveriti definiciju `VER130` da biste proverili da li kompajlirate upotrebom Delphija 5 ili nekom ranijom verzijom. Ovo može da bude korisno ukoliko želite da kompajlirate isti program upotrebom različitih verzija Delphija i načinite manje izmene izvornog koda u svakoj od verzija. Ukoliko želite da dodate nešto specifičnog koda Delphija 5, taj kod možete napisati na sledeći način:

```
{$IDEF VER130}
  // Delphi 5 specific code
{$ENDIF}
```

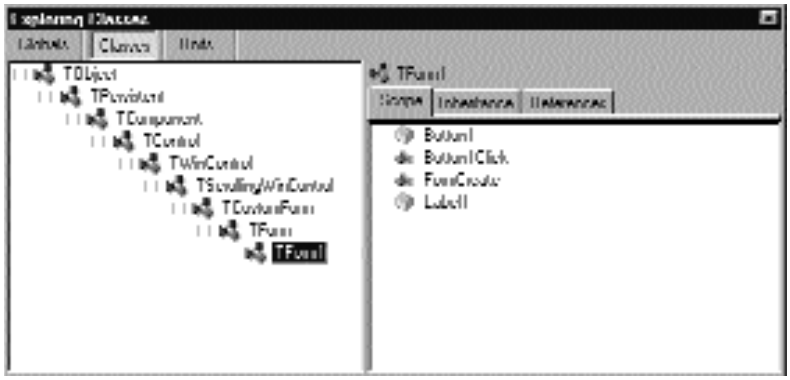
Svaka od ranijih verzija Delphija sadrži specifičnu definiciju tako da možete napisati složeni iskaz da biste obezbedili različita rešenja kodiranja za različite verzije Delphija. Šema nabrajanja počinje od poslednje verzije Pascal kompajlera Borlanda pre pojave Delphija, Borland Pascal with Object verzija 7 i, takođe, sadrži verzije Pascal kompajlera koje sadrži Borland C++ Builder:

- `VER80` for Delphi 1
- `VER90` for Delphi 2
- `VER93` for C++ Builder 1
- `VER100` for Delphi 3
- `VER110` for C++ Builder 3
- `VER120` for Delphi 4
- `VER125` for C++ Builder 4

Pretraživanje projekta

Prethodne verzije Delphija su sadržale Object Browser koji ste mogli da koristite kada je projekat kompajliran, da biste videli hijerarhijsku strukturu klasa, i da biste pogledali simbole i linije izvornog koda kada se na njih referiše. Delphi 5 sadrži sličan, ali poboljšan alat, koji ima novi naziv — Project Explorer. Kao i Code Explorer, automatski se ažurira prilikom unosa, a da ne morate ponovo da kompajlirate projekat.

Project Explorer je od Object Browsera preuzeo osnovnu strukturu klasa (Classes), jedinica (Units) i globala (Globals), ali Vam dopušta da odaberete da li da prikaže samo simbole definisane u Vašem projektu, ili i iz projekta i VCL-a. Možete videti primer samo sa simbolima projekta na slici 1.16.



SLIKA 1.16 *Project Explorer, potpuno ažurirani Object Browser*

Podešavanja ovog Explorera i Code Explorera možete promeniti na strani Explorer okvira za dijalog Environment Options (videti sliku 1.5) ili izborom komande Properties iz kontekst menija Project

Explorera. Neke od kategorija Explorera koje vidite u ovom prozoru su specifične za Project Explorer, a druge se odnose na oba alata.

Dodatni i spoljašnji alati Delphija

Pored IDE-a, kada instalirate Delphi, dobijate i druge, spoljašnje alate. Neki od njih, kao što su Database Desktop, Package Collection Editor (PCE.EXE) i Image Editor (ImagEdit.EXE) su dostupni iz menija Tools IDE-a. Pored toga, izdanje Client/Server sadrži link za SQL Monitor (SqlMon.EXE).

Na kraju, neki od primera programa koje dobijate uz Delphi su zapravo korisni alati koje možete kompajlirati i imati pri ruci. Neke od ovih alata ću razmatrati u knjizi, kako bude bilo potrebno. Ovde su neki od korisnih alata višeg nivoa:

- **WINSIGHT** (WS.EXE) je Windowsov program za presretanje poruka koji se nalazi u direktorijumu `Bin`.
- **DATABASE EXPLORER** se može aktivirati iz Delphi IDE-a ili kao samostalan alat upotrebom programa DBExplor.EXE koji se nalazi u direktorijumu `Bin`.
- **CONVERT** (Convert.EXE) je alat komandne linije koji možete upotrebiti za konvertovanje DFM fajlova u ekvivalentne tekstualne opise i obrnuto.
- **TURBO GREP** (Grep.EXE) je pomoćni program za pretraživanje, mnogo brži od ugrađenog mehanizma Find in Files, ali nije tako lak za korišćenje.
- **TURBO REGISTER SERVER** (TRegSvr.EXE) je alat koji možete upotrebiti za registrovanje ActiveX biblioteka i COM servera. Izvorni kod ovog alata je na raspolaganju u direktorijumu Demos/ActiveX/TRegSvr.
- **RESOURCE EXPLORER** je moćan alat za pregled resursa (ali ne i potpuni resurs-editor) koji možete pronaći u direktorijumu Demos/ResXplor.
- **DELPHI 5 CD** takođe, sadrži odvojenu instalaciju za **Resource Workshop**. To je stari 16-bitni resurs-editor koji se može upotrebiti za Win32 resurs fajlove. Ranije je bio pridružen kompajlerima Borland C++ i Pascal za Windows, i bio je mnogo bolji od standardnog Microsoftovog resurs-editora koji je tada bio na raspolaganju. Mada korisnički interfejs nije izmenjen i mada ne podržava duga imena fajlova, ovaj alat još uvek može da bude koristan za izradu korisničkih ili specijalnih resursa. Alat Vam, takođe, omogućava da pretražujete resurse postojećih izvršnih fajlova. Više informacija o Windowsovim resursima i upotrebi Resource Workshopa ćete pronaći u Poglavlju 19.

Fajlovi koje proizvodi sistem

Delphi proizvodi veliki broj fajlova za svaki projekat i trebalo bi da znate koji su to fajlovi i koji su njihovi nazivi. Postoje, u osnovi, dva elementa koja imaju uticaj na imenovanje fajlova: nazivi koje dodeljujete projektima i njihovim jedinicama, i unapred određene ekstenzije koje koriste Delphi. U tabeli 1.1 je spisak ekstenzija fajlova koje ćete pronaći u direktorijumu u kojem se nalazi Delphi projekat. Tabelom je prikazano kada ili pod kojim uslovima se ti fajlovi kreiraju, kao i njihova važnost za buduće kompajliranje. Ekstenzije koje su nove u Delphiju 5 su naznačene masnim slovima.

Tabela 1.1: Ekstenzije fajlova Delphi projekta

Ekstenzija	Tip fajla i opis	Vreme kreiranja	Potrebni za kompajliranje?
.BMP, .ICO, .CUR	Fajlovi bitmapa, ikona i kursora: standardni Windows fajlovi koji se koriste za čuvanje bitmapiranih slika.	Razvoj: Image Editor	Obično ne, ali su, možda, potrebni u vreme izvršavanja i za dalje izmene.
.BPG	Borland Project Group: fajlovi koje koristi novi višeciljni Project Manager. To je vrsta makefilea.	Razvoj	Potrebni za ponovno kompajliranje odjednom svih projekata grupe.
.BPL	Borland Package Library: DLL koji sadrži VCL komponente koje će koristiti Delphi okruženje prilikom dizajniranja ili koje će koristiti aplikacija prilikom izvršavanja. (Ovi fajlovi su imali .DPL ekstenziju u Delphiju 3.)	Kompilacija: Povezivanje	Prosleđivaćete pakete drugim Delphi programerima i, opciono, krajnjim korisnicima.
.CAB	Microsoft Cabinet kompresovani format fajla koji se koristi za web razvoj u Delphiju. CAB fajlovi mogu sadržati više kompresovanih fajlova.	Kompilacija	Prosleđuju se krajnjim korisnicima
.CFG	Konfiguracijski fajl sa opcijama projekta. Sličan DOF fajlovima.	Razvoj	Potrebni samo ukoliko se koriste specijalne opcije kompajlera.
.DCP	Delphi Component Package: fajl sa informacijama simbola za kod koji se kompajlira u paket. Ne sadrži kompajlirani kod koji se čuva u DCU fajlovima.	Kompilacija	Potrebni kada koristite pakete. Prosleđivaćete ih samo drugim programerima zajedno sa DPL fajlovima.

Tabela 1.1: Ekstenzije fajlova Delphi projekta

Ekstenzija	Tip fajla i opis	Vreme kreiranja	Potrebni za kompajliranje?
.DCU	Delphi Compiled Unit: rezultat kompajliranja Pascal fajla.	Kompilacija	Samo ukoliko izvorni kod nije dostupan. DCU fajlovi za jedinice koje pišete predstavljaju međukorak, te kompajliranje čine bržim.
.DFM	Delphi Form File: binarni fajl sa opisom svojstava formulara (ili modula podataka) i komponenata koje sadrži.	Razvoj	Da. Svaki formular se čuva kako u PAS tako i u DFM fajlu.
.~DF	Rezervna kopija Delphi Form File (DFM).	Razvoj	Ne. Fajl nastaje kada sačuvate novu verziju jedinice koja ima veze sa formularom i uz nju fajl formulara.
.DFN	**Fajl podrške za Integrated Translation Environment (postoji jedan DFN fajl za svaki formular i svaki ciljni jezik).**	**Razvoj (ITE)**	**Da (za ITE). Ovi fajlovi sadrže prevedene stringove koje menjate u Translation Manageru.**
.DLL	Dynamic Link Library: još jedna verzija izvršnog fajla.	Kompilacija: Linkovanje	Pogledati .EXE.
.DOF	Delphi Option File: tekst-fajl sa aktuelnim podešavanjima opcija projekta.	Razvoj	Potrebni samo ukoliko se koriste specijalne opcije kompajlera.
.DPK	Delphi Package: fajl sa izvornim kodom projekta paketa.	Razvoj	Da.
.DPR	Delphi Project File. (Ovaj fajl zapravo sadrži Pascal izvorni kod.)	Razvoj	Da.
.~DP	Rezervna kopija Delphi Project fajla (.DPR)	Razvoj	Ne. Ovaj fajl se automatski generiše kada sačuvate novu verziju fajla projekta.
.DSK	Desktop fajl: sadrži informacije o poziciji Delphi prozora, fajlovima otvorenim u editoru i drugim podešavanjima radne površine.	Razvoj	Ne. Zapravo bi trebalo da ga uklonite kada kopirate projekat u novi direktorijum.
.DSM	Delphi Symbol Module: čuva sve informacije o simbolima pretraživača.	Kompilacija (ali samo ako se koristi opcija Symbols)	Ne. Object Browser koristi ovaj fajl, umesto podataka u memoriji, kada ne možete ponovo kompajlirati projekat.

Ekstenzija	Tip fajla i opis	Vreme kreiranja	Potrebni za kompajliranje?
.DTI	**Design Time Information, koristi ih novi Data Module Designer.**	**Razvoj**	**Ne. Ovaj fajl čuva informacije "u vreme dizajniranja", nije potreban za rezultujući program, ali je veoma važan za svakog programera.**
.EXE	Izvršni fajl: Windows aplikacije koju proizvodite.	Kompilacija: Linkovanje	Ne. Ovo je fajl koji prosleđujete. Sadrži sve kompajlirane jedinice, formulare i resurse.
.HTM	Ili .HTML, za HyperText Markup Language: format fajla koji se koristi za web strane.	Web razvoj za ActiveForm	Ne. Nije potreban za kompajliranje projekta
.LIC	Fajlovi sa licencom koji se odnose na OCX fajl.	ActiveX Wizard i drugi alati	Ne. Neophodan za kontrolu u drugom razvojnom okruženju.
.OBJ	Objekt (kompajlirani) fajl, tipičan za C/C++ svet.	Međukorak prilikom kompajliranja, Object Inspector se ne koristi u Delphiju.	Možda je potreban za spajanje Delphija sa C++ kompajliranim kodom u projektu.
.OCX	OLE Control eXtension: specijalna verzija DLL-a, sadrži ActiveX kontrole ili formulare.	Kompilacija: Linkovanje	Pogledati .EXE.
.PAS	Pascal fajl: izvorni kod Pascal jedinice, bilo da je to jedinica koja ima veze sa formularom, bilo da je samostalna jedinica.	Razvoj	Da.
.čPA	Rezervna kopija Pascal fajla (.PAS)	Razvoj	Ne. Ovaj fajl Delphi automatski generiše kada sačuvate novu verziju izvornog koda.
.RES, .RC	Resurs fajl: binarni fajl pridružen projektu, a obično sadrži ikonu projekta. Možete dodati druge fajlove ovog tipa projektu. Kada kreirate resurs fajlove, možete, takođe, koristiti tekstualni format, .RC.	Razvoj Options dijalog. ITE (Integrated Translation Environment) generiše resurs fajlove sa specijalnim komentarima	Da. Glavni RES fajl aplikacije Delphi ponovo generiše prema informaciji sa strane Aplication okvira za dijalog Project Options.
.RPS	**Translation Repository (deo Integrated Translation Environmenta)**	**Razvoj (ITE)**	**Ne. Potreban za upravljanje prevodima.**
.TLB	Type biblioteka: fajl koji se automatski formira ili ga formira Type Library Editor za OLE server aplikacije.	Razvoj	Ovo je fajl koji je možda potreban za druge OLE programe.

Tabela 1.1: Ekstenzije fajlova Delphi projekta

Ekstenzija	Tip fajla i opis	Vreme kreiranja	Potrebni za kompajliranje?
.TODO	Fajl sa spiskom zadataka, sadrži elemente koji se odnose na ceo projekat.	Razvoj	Ne. Ovaj fajl sadrži beleške programera.
.UDL	Microsoft Data Link	Razvoj	Koristi ga ADO da bi se referisao na provajdera podataka. Slično kao alijas u BDE svetu (videti Poglavlje 12).

Pored fajlova koji se generišu prilikom razvoja projekta u Delphiju, postoje i mnogi drugi koje generiše i koristi IDE. U tabeli 1.2 sam dao kratku listu ekstenzija koje nije na odmet znati. Većina ovih fajlova je u ekskluzivnom i nedokumentovanom formatu, te je malo toga što sa njima možete učiniti.

Tabela 1.2: Odabrane Delphi IDE ekstenzije fajlova prilagođavanja

Ekstenzija	Tip fajla
.DCI	Delphi Code Templates
.DRO	Delphi Object (Može se menjati komandom Tools➥Repository.)
.DMT	Delphi Menu Templates
.DBI	Database Explorer Information
.DEM	Delphi Edit Mask (fajlovi sa specifičnim formatima za zemlje)
.DCT	Delphi Component Templates
.DST	Fajl sa podešavanjima radne površine (po jedan za svako podešavanje koje definišete)

Prikazivanje fajlova sa izvornim kodom

Upravo sam nabrojao fajlove koji su u vezi sa razvojem aplikacije Delphi, ali želim da posvetim malo više pažnje objašnjenju njihovih formata. Osnovni Delphi fajlovi su Pascal fajlovi sa izvornim kodom, koji su zapravo ASCII tekst fajlovi. Masna slova, kurziv i obojeni tekst koji vidite u editoru, zavise od označavanja sintakse, ali se ne čuvaju u okviru fajla. Nemaju nikakvu vrednost jer postoji jedan fajl za sav kod formulara, a ne za male fragmente koda.

NAPOMENA

U listinzima u knjizi sam pokušao da koristim masna slova kao što to koristi editor za ključne reči, i kurziv za stringove i komentare. ■

Za formular, Pascal fajl sadrži deklaraciju klase formulara i izvorni kod rukovanja događajima. Vrednosti svojstava koje određujete u Object Inspectoru se čuvaju u zasebnom fajlu sa opisom (sa ekstenzijom .DFM). Jedini izuzetak je svojstvo `Name`, koje se koristi u deklaraciji formulara da bi se referisalo na komponente formulara.

DFM fajl je binarni fajl i u Delphiju 5 se može sačuvati bilo kao tekstualni fajl, bilo u tradicionalnom formatu Windows Resource. Možete odrediti format koji želite da koristite za nove projekte na strani References okvira za dijalog Environment Options, i možete menjati format za pojedine formulare komandom Text DFM iz kontekst menija formulara. Tekstualni editor može pročitati samo tekstualne fajlove. Ipak, možete učitati DFM falove oba tipa u Delphi editor, koji će, ukoliko je potrebno, prvo konvertovati fajlove u tekstualni opis. Najjednostavniji način za otvaranje tekstualnog opisa formulara (u bilo kom da je formatu) je da odaberete komandu View as Text iz kontekst menija Form Designera. Na ovaj način ćete zatvoriti formular, čuvajući ga ukoliko je potrebno, i otvoriti DFM fajl u editoru. Kasnije se možete vratiti na formular koristeći komandu View as Form iz kontekst menija prozora editora.

Vi, zapravo, možete editovati tekstualni opis formulara, mada to treba učiniti veoma pažljivo. Čim sačuvate fajl, biće pretvoren u binarni fajl. Ukoliko ste načinili nepravilne izmene, kompajliranje će se zaustaviti uz poruku o grešci i biće potrebno da ispravite sadržaj DFM fajla Project Explorer nego što ponovo budete mogli da otvorite formular. Zbog toga ne bi trebalo da ručno pokušavate da izmenite tekstualni opis formulara, sve dok ne steknete dobro znanje o Delphi programiranju.

NAPOMENA

U knjizi ću Vam često prikazivati delove DFM fajlova. U većini delova ću prikazivati samo najrelevantnije komponente ili svojstva; uopšte, ja ću ukloniti svojstva koja se odnose na poziciju, binarne vrednosti i druge linije koje daju malo korisnih informacija. ■

Uz dva fajla koja opisuju formular (PAS i DFM), treći fajl je srž za ponovnu izgradnju aplikacije. To je Delphi projekt fajl (Delphi project file — DPR), koji predstavlja još jedan Pascal fajl sa izvornim kodom. Ovaj fajl se automatski formira i retko ćete imati potrebu da ga ručno menjate. Ovaj fajl možete prikazati komandom View➥Project Source.

Neki drugi, manje važni fajlovi koje proizvodi IDE, koriste strukturu Windows INI fajlova, u kojima je svaka sekcija označena nazivom koji se smešta unutar uglastih zagrada. Na primer, ovo je deo opcionog fajla (DOF):

```
[Compiler]
A=1
B=0
ShowHints=l
Showwarnings=1

[Linker]
MinStackSize=16384
MaxStackSize=1048576
ImageBase=4194304

[Parameters]
RunParams=
HostApplication=
```

Istu strukturu koriste Desktop fajlovi (DSK), u kojima se čuva status Delphi IDE-a za određeni projekat, prikazujući poziciju svakog od prozora. Evo malog isečka:

```
[MainWindow]
Create=1
Visible=1
State=0
Lett=2
Top=0
Width=800
Height=97
```

NAPOMENA

Mnogo informacija koje se odnose na status okruženja Delphija se čuva u Windows Registryju, kao i u DSK i drugim fajlovima. Ja sam već naznačio nekoliko specijalnih nedokumentovanih stavki Registryja koje možete upotrebiti da biste aktivirali specifične funkcije. Trebalo bi da istražite sekciju Registryja `HKEY_CURENT_USER/Software/Borland/Delphi/5.0` da biste proučili sve vrednosti Delphi IDE-a (uključujući sve koje možete izmeniti upotrebom okvira za dijalog Project Options i Environment Options, kao i mnoge druge). ■

Object Repository

Delphi sadrži nekoliko komandi menija koje možete upotrebiti za kreiranje novog formulara, nove aplikacije, novog modula podataka, nove komponente i tako dalje. Ove komande se nalaze u meniju File i drugim menijima. Šta se dešava ukoliko jednostavno odabere File➥New? Delphi otvara Object Repository koji se koristi za kreiranje novih elemenata bilo koje vrste: formulara, aplikacija, modula podataka, biblioteka, komponenata, objekata automatizacije i drugih. Okvir za dijalog New (prikazan na slici 1.17) sadrži više strana na kojima se nalaze svi novi elementi koje možete kreirati, postojeći formulari i projekti koji se čuvaju u Repositoryju, Delphi čarobnjaci i formulari aktuelnog projekta (za vizuelno nasleđivanje formulara). Strane i stavke u ovom okviru za dijalog sa stranicama zavise od određene verzije Delphija, tako da ih ovde neću nabrojati.

SLIKA 1.17 *Prva strana okvira za dijalog New, opšte poznata kao Object Repository*

SAVET

Object Repository sadrži kontekst meni koji omogućava da sortirate elemente na različite načine (po nazivima, autoru, datumu ili opisu) i da ih prikažete u različitim pogledima (velike ikone, male ikone, spisak ili detaljan spisak). Pogled Details Vam daje opis, autora i datum alata, informacije koje su naročito važne kada prikazujete čarobnjake, projekte ili formulare koje ste dodali u Repository. ■

Najjednostavniji način da prilagodite Object Repository je da dodate nove projekte, formulare i module podataka kao šablone. Takođe, možete dodati i nove strane i urediti elemente na nekima od njih (ne uključujući strane New i aktuelne projekte). Dodavanje novog šablona Delphi Object Repository je jednostavno koliko i upotreba postojećeg šablona za izradu aplikacije. Kada imate aplikaciju koja funkcioniše, a želite da je upotrebite kao početnu tačku za budući razvoj sličnih programa, možete sačuvati trenutni status u šablonu koji možete kasnije koristiti. Jednostavno upotrebite komandu Project➥Add to Repository i popunite okvir za dijalog.

Kao što možete dodati šablone projekta u Object Repository, možete, takođe, dodati nove šablone formulara. Jednostavno pronađite formular koji želite da dodate i odaberite komandu Add to Repository iz kontekst menija. Zatim naznačite naslov, opis, autora, stranu i ikonu u okviru za dijalog.

Imajte na umu da, kada kopirate projekat ili formular iz šablona u Repository, a zatim kopirate u drugi direktorijum, Vi jednostavno izvršavate operaciju kopiranja i smeštanja. Ovo se ne razlikuje mnogo od ručnog kopiranja fajlova.

PRAZAN ŠABLON PROJEKTA

Kada započnete prazan projekat, takođe se automatski otvara prazan formular. Ukoliko želite da novi projekat bazirate na nekom od formulara ili čarobnjaka (Wizards), ovo nije ono što želite. Da biste rešili problem, možete dodati šablon Empty Project (prazan projekat) u Gallery.

Koraci koji su neophodni su sasvim jednostavni.

1. Kreirajte, kao i obično, novi projekat.
2. Uklonite jedini formular iz projekta.
3. Dodajte ovaj projekat šablonima, dodeljujući mu naziv Empty Project.

Kada odaberte ovaj projekat iz okvira za dijalog Object Repository, imate dve prednosti. Imate projekat bez formulara i možete odabrati direktorijum u koji će se kopirati fajlovi šablona projekta. Postoji i nedostatak — morate zapamtiti da upotrebite komandu File➥Save Project As da biste projektu dodelili novi naziv, jer čuvanje projekta na bilo koji drugi način automatski koristi unapred određeni naziv šablona.

Da biste dalje prilagodili Repository, možete upotrebiti komandu Tools➥Repository. Na ovaj način ćete otvoriti okvir za dijalog Object Repository koji možete upotrebiti da elemente premestite na neku drugu stranu okvira za dijalog, da dodate nove elemente ili da uklonite postojeće. Možete čak dodati nove strane, promeniti naziv strani ili je ukloniti, ili promeniti njihov redosled. Važan element pri izmeni okvira za dijalog Object Repository je upotreba unapred određenih vrednosti.

- Upotrebite kvadratić za potvrdu New Form koji se nalazi ispod liste objekata, da biste naznačili formular koji treba koristiti prilikom kreiranja novih formulara (File➥New Form).
- Kvadratić za potvrdu Main Form označava koji tip formulara se koristi prilikom kreiranja glavnog formulara aplikacije (File➥New Application) kada nije odabran nijedan novi projekat.
- Kvadratić za potvrdu New Project, dostupan kada odaberete projekat, označava unapred određeni projekat koji će Delphi koristiti kada pozovete komandu File➥New Application.

Samo jedan formular i samo jedan projekat u okviru za dijalog Object Repository može imati svako od ova tri svojstva označena specijalnim simbolom koji se pojavljuje iznad ikone. Ukoliko nijedan projekat nije označen kao New Project, Delphi kreira unapred određeni projekat na osnovu formulara označenog kao Main Form. Ukoliko nijedan formular nije označen kao glavni fomular, Delphi kreira unapred određeni projekat na osnovu praznog formulara. Kada radite sa okvirom za dijalog Object Repository, radite sa formularima i modulima sačuvanim u poddirektorijumu `OBJREPOS` glavnog Delphi direktorijuma. Istovremeno, ukoliko direktno koristite formular ili bilo koji drugi objekat bez prethodnog kopiranja, tada ćete dobiti neke fajlove Vašeg projekta u ovom direktorijumu. Veoma je važno shvatiti kako Repository funkcioniše, jer ukoliko želite da izmenite projekat ili objekat sačuvan u Repositoryju, najbolji pristup je da operišete sa originalnim fajlovima i da ne kopirate podatke u i iz Repositoryja.

INSTALIRANJE NOVIH DLL ČAROBNJAKA

U osnovi, novi čarobnjaci dolaze u dva različita oblika: mogu da budu deo komponenata ili paketa, ili se mogu distribuirati kao nezavisni DLL-ovi. U prvom slučaju, instaliraju se na isti način kao što se instaliraju komponente ili paketi. Kada dobijete samostalni DLL, potrebno je da dodate naziv DLL-a u `Windows Registry` pod ključem `Software\Borland\Delphi\5.0\Experts`. Jednostavno dodajte novi string ključ pod ovaj ključ, odaberite naziv koji želite (naziv nije zapravo važan), i upotrebite kao tekst putanju i naziv fajla DLL čarobnjaka. Možete pogledati elemente koji već postoje pod ključem `Experts` da biste videli kako da unesete putanju.

Šta je sledeće?

Ovo poglavlje sadrži pregled novih i naprednijih funkcija programskog okruženja Delphi 5, uključujući brojne savete i sugestije o nekim manje poznatim funkcijama koje su već bile na raspolaganju u prethodnim verzijama Delphija. Nisam naveo opise korak-po-korak za IDE, delimično zbog toga što je, uopšte uzev, lakše započeti upotrebu Delphija nego što je lako pročitati kako ga koristiti. Takođe, postoji detaljan Help fajl koji opisuje okruženje i razvoj novog jednostavnog projekta, a možda već posedujete nekakvo iskustvo u korišćenju prethodnih verzija Delphija ili sličnih razvojnih okruženja. Mi nismo završili upoznavanje novih funkcija Delphi 5 IDE-a. Razmatraću novi Data Module Designer u Poglavlju 10, nove funkcije debagovanja u Poglavlju 18 i TeamSource i Integrated Translation Environment u Poglavlju 19. Ali, mi smo sada spremni da naredna tri poglavlja posvetimo jeziku Object Pascal i VCL biblioteci. Zatim ćemo se, u Delu II, pozabaviti korisničkim interfejsom aplikacija i upotrebom komponenata koje su na raspolaganju u Delphiju.

POGLAVLJE

2

Objektno orijentisano programiranje u Delphiju

VEĆINA SAVREMENIH PROGRAMSKIH JEZIKA PODRŽAVA OBJEKTNO ORIJENTISANO PROGRAMIRANJE (OOP). OOP JEZICI SU ZASNOVANI NA TRI FUNDAMENTALNA KONCEPTA: ENKAPSULACIJI (OBIČNO JE IMPLEMENTIRANA KOD KLASA), NASLEĐIVANJU I POLIMORFIZMU (ILI KASNOM POVEZIVANJU).

Delphi aplikacije možete napisati čak i bez znanja detalja Object Pascala. Kada kreirate novi formular, dodajete nove komponente i rukujete događajima. Delphi automatski priprema veći deo odgovarajućeg koda za Vas. Međutim, poznavanje detalja jezika i njegove implementacije će Vam pomoći da bolje razumete šta Delphi radi, i da u potpunosti savladate jezik.

Jedno poglavlje nema dovoljno prostora za potpuni uvod u principe objektno orijentisanog programiranja i jezik Object Pascal. Umesto toga, ja ću istaći ključne OOP funkcije jezika i pokazaću kakve veze imaju sa svakodnevnim programiranjem. Iako nemate detaljno znanje o OOP-u, poglavlje će Vam predstaviti svaki od ključnih koncepata tako da nećete morati da se referišete na druge izvore.

NAPOMENA

Ukoliko ne poznajete osnove jezika Pascal (koji nisu objašnjeni u ovoj knjizi), možete pogledati elektronsku verziju teksta "Osnove Pascala" (Essential Pascal) na adresi `www.marcocantu.com`. Jezik se nije bitno promenio od Delphija 4 do Delphija 5. ■

Uvod u klase i objekte

Klasa i *objekat* su dva termina koja se obično koriste u Object Pascalu i drugim OOP jezicima. Ipak, s obzirom na to da se često pogrešno koriste, postarajmo se da se saglasimo oko njihovih definicija. *Klasa* (class) je korisnički definisan tip podataka koji ima stanje (svoju reprezentaciju) i neke operacije (svoje ponašanje). Klasa sadrži neke interne podatke i neke metode, u formi procedura i funkcija, i obično opisuje generičke karakteristike i ponašanje velikog broja sličnih objekata.

Objekat (object) je instanca klase, ili promenljiva tipa podataka definisanog klasom. Objekti su *stvarni* entiteti. Kada se program izvršava, objekti zauzimaju deo memorije za njihove interne reprezentacije. Veza između objekata i klasa je jednaka vezi između promenljive i tipa podataka.

Da biste deklarisali tip podataka klase u Object Pascalu sa nekim lokalnim poljima podataka i nekim metodima, upotrebite sledeću sintaksu:

```
type
  TDate = class
    Month, Day, Year: Integer;
    procedure SetValue (a, d, y: Integer);
    function LeapYear: Boolean;
end;
```

Funkcija i procedura definisane u kodu iznad trebalo bi da u potpunosti budu definisane u implementacionom delu iste jedinice, uključujući i deklaraciju klase. Možete dopustiti Delphiju da generiše kostur definicije metoda upotrebom funkcije editora Class Completion (jednostavno pritisnite kombinaciju tastera Ctrl+C dok se kursor nalazi unutar definicije klase). Možete naznačiti da su metodi deo klase `TDate` po prefiksu naziva klase (koristeći tačku između) kao u sledećem kodu:

```
procedure TDate.SetValue(m, d, y: Integer);
begin
  Month := m;
  Day := d;
  Year := y;
end;

function TDate.LeapYear: Boolean;
begin
  // call IsLeapYear in SysUtils.pas
  Result := IsLeapYear (Year);
end;
```

SAVET

U Delphiju je usvojeno da se koristi slovo T kao prefiks u nazivu svake klase koju napišete i bilo kog drugog tipa (T je skraćenica od Type). To je samo konvencija — za kompajler, T je slovo kao i bilo koje drugo slovo — ali je toliko uobičajeno da ćete njegovom upotrebom učiniti kod razumljivijim. Ja ću pokušati da se držim ove konvencije u knjizi. ■

Kada je klasa definisana, možemo kreirati objekat i koristiti ga na sledeći način:

```
var
  ADay:  TDate;
begin
  // create
  ADay := TDate.Create;
  // use
  ADay.Setvalue(1,1,2000);
  if ADay.LeapYear then
    ShowMessage ('Leap year: ' + IntToStr (ADay.Year));
  // destroy
  ADay.Free;
end;
```

Notacija koja se koristi nije neobična, ali je veoma moćna. Možete napisati složenu funkciju (kao što je `LeapYear`), a zatim dobiti njenu vrednost za svaki objekat `TDate` kao da je to primitivni tip podataka. Primetite da je izraz `ADay.LeapYear` sličan `ADay.Year`, mada je prvo poziv funkcije, a drugo direktan pristup podacima. Kao što ćemo videti u narednom poglavlju, notacija koju koristi Object Pascal, da bi se pristupilo svojstvima, je ponovo ista.

Model Object Reference u Delphiju

U nekim OOP jezicima, deklarisanje promenljive tipa klase kreira instancu te klase. Object Pascal se, umesto toga, zasniva na *objektnom referentnom modelu*. Ideja je da svaka promenljiva tipa klase, kao što je `ADay` u fragmentu koda iznad, ne sadrži vrednost objekta. Umesto toga sadrži referencu, ili *pokazivač*, kojim se označava lokacija u memoriji gde se objekat čuva.

NAPOMENA

Objektni referentni model je moćan i lakše ga je koristiti od ostalih modela. Drugi OOP jezici koriste slične modele, naročito Eifel i Java. Po mom mišljenju, prihvatanje ovih modela je bila jedna od najboljih odluka razvojnog tima Delphija. ■

Jedini problem ovakvog pristupa je da kada deklarišete promenljivu, Vi ne kreirate objekat u memoriji; Vi samo rezervišete lokaciju memorije za referencu na objekat. Instance objekta se moraju ručno kreirati, bar za objekte klase koju definišete. Instance komponente koje smeštate na formular, Delphi sam izgrađuje.

Da biste kreirali instancu objekta, možemo pozvati metod `Create`, koji predstavlja konstruktor. Kao što ste mogli da vidite u poslednjem fragmentu koda, konstruktor se odnosi na klasu, a ne na objekat. Odakle dolazi metod `Create`? To je konstruktor klase `TObject`, iz koje ga nasleđuju sve ostale klase. Kada jednom kreirate objekat, i kada ste završili sa korišćenjem objekta, potrebno je da ga uklonite da biste izbegli popunjavanje memorije objektima koji Vam više nisu potrebni, što dovodi do stanja koje se naziva "iscrpljenje memorije" (memory leak). Oslobađanje memorije se može postići pozivom metoda `Free` (koji predstavlja još jedan metod klase `TObject`), kao što je pokazano u prethodnom listingu. Sve dok kreirate objekte kada su Vam potrebni, i sve dok ih uklanjate kada Vam nisu potrebni, objektni referentni model će funkcionisati bez greške.

Private, Protected i Public

Klasa može sadržati bilo koju količinu podataka i bilo koji broj metoda. Ipak, za dobar objektno orijentisani pristup, podaci bi trebalo da budu sakriveni, ili enkapsulirani (encapsulated), unutar klase koja ih koristi. Kada pristupate datumu, na primer, nema nikakvog smisla menjati samu vrednost datuma. Zapravo, promena vrednosti dana može kao rezultat dati netačan datum, recimo 30. februar. Upotreba metoda za pristup unutrašnjoj reprezentaciji objekta smanjuje rizik generisanja grešaka, jer metodi mogu proveriti da li je datum valjan i odbiti promenu datuma ukoliko nije. Enkapsulacija je važna jer omogućava onome ko piše klasu, da promeni internu reprezentaciju u nekoj budućoj verziji.

Koncept enkapsulacije je veoma jednostavan: zamislite klasu kao "crnu kutiju" čiji je jedan mali deo vidljiv. Vidljivi deo, koji se naziva *interfejs klase*, omogućava ostalim delovima programa pristup i upotrebu objekata klase. Ipak, kada koristite objekte, većina njihovog koda je sakrivena. Retko ćete znati koje interne podatke objekat ima, i obično nećete imati način da direktno pristupite podacima. Naravno, pretpostavlja se da ćete Vi koristiti metode da biste pristupili podacima koji su zaštićeni od neautorizovanog pristupa. Ovo je objektno orijentisani pristup klasičnom konceptu programiranja koji je poznat kao sakrivanje informacija (information hiding).

Object Pascal sadrži tri identifikatora pristupa: `private`, `protected` i `public`. Četvrti, `published`, ćemo razmatrati u narednom poglavlju. Evo tri osnovna:

- Direktiva `private` označava polja i metode klase kojima se ne može pristupiti van jedinice (fajla izvornog koda) koja deklariše klasu.
- Direktiva `public` označava polja i metode kojima se može slobodno pristupati iz bilo kog dela programa kao i iz jedinice u kojoj su definisani.

- Direktiva `protected` se koristi da označi metode i polja sa ograničenom vidljivošću. Samo aktuelna klasa i njene potklase mogu pristupiti elementima `protected`. Ponovo ćemo razmatrati ovu ključnu reč u odeljku "`Protected` polja i enkapsulacija".

Uopšte uzev, polja klase bi trebalo da budu `private`; metodi su obično `public`. Ipak, to nije uvek slučaj. Metodi mogu da budu `private` ili `protected` ukoliko su potrebni samo interno, da bi obavili neka delimična izračunavanja. Polja mogu da budu `protected` ili `public` kada želite lak i direktan pristup i kada ste prilično sigurni da se njihova definicija tipa neće menjati.

SAVET

Umesto da imate `public` polja, trebalo bi da koristite svojstva, kao što ćemo detaljno videti u narednom poglavlju. Svojstva su proširenje mehanizma enkapsulacije drugih OOP jezika i veoma su važna u Object Pascalu. ■

Specifikatori pristupa samo ograničavaju kod van jedinice prilikom pristupa određenim članovima klasa koji su deklarisani u odeljku interfejs Vaše jedinice. To znači da ukoliko su dve klase u okviru iste jedinice, tada ne postoji zaštita njihovih privatnih polja. Samo ukoliko smestite klasu u deo jedinice interfejs, moći ćete da ograničite vidljivost klasa i funkcija u drugim jedinicama na javne metode i polja klase.

Razmislite o ovoj novoj verziji klase `TDate`.

```
type
  TDate = class
  private
    Month, Day, Year: Integer;
  public
    procedure SetValue (m, d, y: Integer);
    function LeapYear: Boolean;
    function GetText: string;
    procedure Increase;
  end;
```

U ovoj verziji polja su deklarisana kao `private` i dodati su neki metodi. Prvi, `GetText`, je funkcija koja kao rezultat vraća string koji sadrži datum. Možda pomišljate da dodate i druge funkcije, recimo `GetDay`, `GetMonth` i `GetYear`, koje će jednostavno kao rezultat dati odgovarajuće `private` podatke, ali slične funkcije direktnog pristupa podacima nisu uvek potrebne. Obezbeđivanje funkcija za pristup svakom polju može umanjiti enkapsulaciju i otežati izmene interne implementacije klase. Funkcije pristupa treba obezbediti samo ukoliko su deo logičkog interfejsa klase koju implementirate.

Drugi novi metod je procedura `Increase`, kojom se datum povećava na naredni dan. Ovo je daleko od jednostavnosti, jer morate uzeti u obzir različiti broj dana u mesecima, kao i prestupne godine. Ono što ću učiniti da bih olakšao pisanje koda, je promena interne implementacije klase da bih mogao da upotrebim Delphijev tip `TDateTime` za internu implementaciju. Klasa će se promeniti u

```
type
  TDate = class
  private
    fDate: TDateTime;
  public
  procedure SetValue (in, d, y: Integer);
  function LeapYear: Boolean;
  function GetText: string;
  procedure Increase;
end;
```

Primetićete da, pošto je jedina promena u `private` delu klase, nećete morati da menjate bilo koji Vaš postojeći program da biste mogli da je koristite. To je prednost enkapsulacije!

NAPOMENA

Tip `TDateTime` predstavlja zapravo broj u pokretnom zarezu. Integralni deo broja označava datume od 12/30/1899, što je isti početni datum koji koriste OLE Automation i Microsoft aplikacije. (Koristite negativne brojeve da biste naznačili ranije godine.) Decimalni deo označava vreme kao razlomak. Na primer, vrednost `3.75` označava drugi januar 1900, i 6 sati po podne (tri četvrtine dana). Da biste dodali ili oduzeli dane, možete jednostavno dodati i oduzeti broj dana, što je mnogo lakše nego dodavanje dana u reprezentaciji dan/mesec/godina. ■

Enkapsulacija i formulari

Jedna od ključnih ideja enkapsulacije je smanjivanje globalnih promenljivih koje koristi program. Globalnim promenljivama se može pristupiti iz bilo kog dela programa. Zbog toga promena vrednosti globalne promenljive ima uticaj u celom programu. S druge strane, kada promenite reprezentaciju polja klase, potrebno je da samo promenite kod nekih metoda klase i ništa više. Dakle, možemo reći da se sakrivanje informacija odnosi na enkapsulaciju izmena (encapsulation changes).

Dopustite mi da Vam primerom razjasnim ideju. Kada imate program sa više formulara, možete neke podatke učiniti dostupnim na svakom formularu tako što ćete ih deklarisati kao globalne promenljive u interfejs delu jedinice u jednom od formulara.

```
var
  Forml:  TForml;
  nClicks: Integer;
```

Ovo će funkcionisati, ali postoje dva problema. Prvo, podaci nisu vezani za određene instance formulara, već za ceo program. Ukoliko kreirate dva formulara istog tipa, oni će deliti podatke. Ukoliko želite da svaki od formulara ima svoju kopiju podataka, jedino rešenje je da ga dodate klasi formulara.

```
type
  TForml = class(TForm)
  public
    nClicks: Integer;
  end;
```

Drugi problem je u tome da ukoliko definišete podatke kao globalne promenljive ili kao public polje formulara, tada nećete moći da u budućnosti izmenite implementaciju, a da time ne utičete na kod koji koristi podatke. Na primer, ukoliko je potrebno samo da pročitate aktuelnu vrednost sa ostalih formulara, možete deklarisati podatke kao private i obezbediti metod koji će pročitati vrednost.

```
type
  TForml = class(Ttorm)
  public
    function GetClicks: Integer;
  private
    nClicks: Integer;
  end;

function TForml.GetClickS Integer;
begin
  Result := nClicks;
end;
```

Još bolje rešenje je dodati svojstvo formularu što ćemo videti u narednom poglavlju.

Ključna reč Self

Videli smo da su metodi slični procedurama i funkcijama. Suštinska razlika je u tome da metodi sadrže implicitni parametar, koji predstavlja referencu na aktuelni objekat. U okviru metoda Vi možete da se referišete na taj parametar — aktuelni objekat — koristeći ključnu reč `Self`. Ovaj dodatni sakriveni parametar je potreban kada kreirate nekoliko objekata iste klase, tako da svaki put kada primenjujete metod na neki od objekata, metod će funkcionisati samo nad sopstvenim podacima i neće uticati na ostale slične objekte.

Na primer, u metodu `SetValue` klase `TDate`, koja je prikazana ranije, mi jednostavno koristimo `Month`, `Year` i `Day` da bismo se referisali na polja aktuelnog objekta, nešto što možete zapisati kao

```
Self.Month := m;
Self.Day := d;
```

Ovako, zapravo, Delphi kompajler prevodi kod, a *ne* kako bi trebalo da ga Vi napišete. Ključna reč `Self` je fundamentalna konstrukcija jezika, koju koristi kompajler, ali je programeri povremeno koriste da bi rešili konflikte sa nazivima i da bi učinili kod čitljivijim. (Jezici C++ i Java imaju sličnu kostrukciju koja se zasniva na ključnoj reči `this`.)

Sve što, zapravo, treba da znate o ključnoj reči `Self`, je to da se tehnička implementacija poziva metodu razlikuje od poziva generičkoj rutini. Metodi imaju dodatni sakriveni parametar, `Self`. S obzirom na to da se sve ovo dešava iza scene, nije potrebno da za sada znate kako funkcioniše `Self`.

SAVET

Ukoliko pogledate definiciju tipa podataka `TMethod` u VCL-u, primetićete da je to slog sa poljima `Code` i `Data`. Prvo polje predstavlja pokazivač na adresu funkcije u memoriji, dok drugo predstavlja vrednost parametra `Self` koji se koristi prilikom poziva adrese funkcije. U narednom poglavlju ćemo razmatrati pokazivače metoda. ■

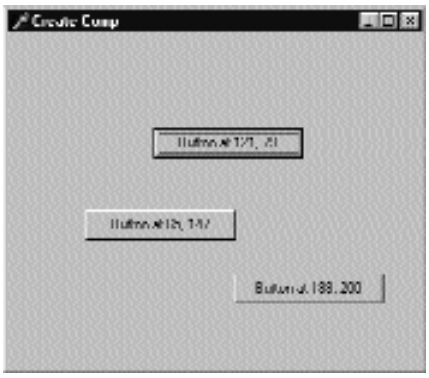
Dinamičko kreiranje komponenata

U Delphiju se ključna reč `Self` često koristi kada je potrebno da se eksplicitno referišete na aktuelni formular u jednom od njegovih metoda. Tipičan primer je kreiranje komponente u vreme izvršavanja, kada morate da prosledite vlasnika komponente njegovom konstruktoru `Create` i dodelite istu vrednost njegovom svojstvu `Parent`. (Razlika između svojstava `Owner` i `Parent` se razmatra u narednom poglavlju.) U oba slučaja je potrebno da obezbedite aktuelni formular kao parametar ili vrednost, a najbolji način da to uradite je ključna reč `Self`.

Da bih Vam demonstrirao ovaj tip koda, napisao sam primer CreateC (naziv je potekao od *Create Component*). Ovaj program sadrži jednostvan formular bez komponenata i hendler za događaj `OnMouseDown`. Koristio sam `OnMouseDown` jer događaj prima parametar kao poziciju gde ste kliknuli mišem (za razliku od događaja `OnClick`). Ova informacija mi je bila potrebna da bih kreirao kontrolu na toj poziciji. Evo koda za metod:

```
procedure TForml. FormMouseDown (Sender: Tobject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
var
  Btn: TButton;
begin
  Btn := TButton.Create (Self);
  Btn.Parent : Self;
  Btn.Left := X;
  Btn.Top := Y;
  Btn.Width := Btn.Width + 50;
  Btn.Caption := Format ('Button at %d, %d', [X, Y]);
end;
```

Efekat ovog koda je u tome da kreira kontrole na mestima gde kliknete mišem, a naslov kontrola predstavlja tačnu poziciju, kao što možete videti na slici 2.1. U prethodnom kodu primetićete upotrebu ključne reči `Self`, kao parametra metoda `Create` i kao vrednost svojstva `Parent`.



SLIKA 2.1 *Rezultat primera CreateC, kojim se kreiraju kontrole u vreme izvršavanja*

Uobičajeno je napisati kod kao kod prethodnog metoda koristeći iskaz `with`, kao u narednom listingu:

```
procedure TForm1.FormMouseDown (Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  with TButton.Create (Self) do
  begin
    Parent := Self;
    Left := X;
    Top := Y;
    Width := Width + 50;
    Caption := Format ('Button in %d %d', [X, Y]);
  end;
end;
```

SAVET

Kada pišete proceduru kao što je kod koji ste upravo videli, možda ćete biti u iskušenju da upotrebite promenljivu `Form1` umesto `Self`. U ovom primeru izmena ne bi donela nikakvu razliku, ali ukoliko postoji više instanci formulara, upotreba `Form1` bi, zapravo, bila greška. Ukoliko se promenljiva `Form1` odnosi na prvi formular tog tipa koji se kreira, kada kliknete neki drugi formular istog tipa, nova kontrola bi se uvek prikazala na prvom formularu. Owner i Parent bi bio `Form1`, a ne formular na koji je korisnik kliknuo. Uopšte, referisanje na određenu instancu klase, kada je potreban aktuelni objekat, je loša OOP praksa. ■

Konstruktori

Da bismo postavili memoriju za objekat, pozvaćemo metod `Create`. To je *konstruktor*, specijalni metod koji možete upotrebiti nad klasom da biste postavili memoriju za instancu klase. Konstruktor će vratiti instancu koja se može dodeliti promenljivoj, koja će čuvati objekat i koju možete kasnije koristiti. Unapred određeni konstruktor `TObject.Create` inicijalizuje sve podatke nove instance na nulu.

Ukoliko želite da Vaša instanca podataka započne ne-nultom vrednošću, tada je potrebno da napišete svoj konstruktor koji će to učiniti. Novi konstruktor možete nazvati `Create`, ili mu možete dodeliti neki drugi naziv: jednostavno upotrebite ključnu reč `constructor` ispred naziva. Primetićete da u tom slučaju nije potrebno pozvati `TObject.Create`: svaki konstruktor automatski može da postavi memoriju za instancu objekta jednostavnom upotrebom ovog specijalnog metoda na odgovarajuću klasu.

Osnovni razlog dodavanja sopstvenog konstruktora klasi je inicijalizacija podataka. Ukoliko objekte kreirate tako da ih ne inicijalizujete, kasniji pozivi metoda mogu kao rezultat imati čudno ponašanje ili izazvati greške prilikom izvršavanja. Umesto da čekate da se greške jave, trebalo bi da koristite preventivne tehnike da biste ih izbegli. Jedna od takvih tehnika je dosledna upotreba konstruktora za inicijalizaciju podataka objekata. Na primer, moramo pozvati proceduru `SetValue` klase `TDate` posle kreiranja objekta. Kao alternativu možemo obezbediti prilagođeni konstruktor koji kreira objekat i dodeljuje mu početnu vrednost.

Mada, uopšte uzev, možete dodeliti bilo koji naziv konstruktoru, imajte na umu da, ukoliko koristite naziv drugačiji od `Create`, konstruktor `Create` osnovne klase `TObject` će i dalje biti na raspolaganju. Ukoliko pišete i razvijate kod koji će drugi koristiti, programer koji poziva unapred određeni konstruktor, može zaobići kod koji ste Vi obezbedili. Definisanjem konstruktora `Create` sa nekoliko parametara Vi ćete zameniti unapred određenu definiciju novom koja će biti

obavezna. Ovo je moguće za generičke klase, ali takav pristup treba izbegavati za korisničke komponente. Kao što ćemo videti u Poglavlju 3, kada nasleđujete od `TComponent`, potrebno je da zaobiđete unapred određeni konstruktor `Create` jednim parametrom i potrebno je da izbegnete onemogućavanje.

Na isti način na koji klasa može imati korisnički konstruktor, može imati i korisnički destruktor, metod koji se deklariše ključnom reči `destructor` i naziva `Destroy`, koji obavlja oslobađanje resursa pre nego što se objekat ukloni. Baš kao što poziv konstruktoru obavlja postavljanje memorije, destruktor oslobađa memoriju. Destruktori su potrebni samo za objekte koji zahtevaju resurse u svojim konstruktorima ili tokom njihovog života.

Umesto da `Destroy` direktno pozivate, program bi trebalo da pozove `Free`, koji poziva `Destroy` samo ukoliko objekat postoji — to jest, ukoliko nije `nil`. Imajte ipak na umu da pozivanje `Free` ne dodeljuje automatski `nil` objektu; to morate sami učiniti! Razlog za to je što objekat ne zna koje se promenljive mogu odnositi na njega, tako da ne postoji način da im dodeli `nil`.

SAVET

Delphi 5 je konačno predstavio jednostavnu proceduru `FreeAndNil` koju možete upotrebiti da biste oslobodili objekat i istovremeno dodeliti njegovoj referenci `nil`. Jednostavno pozovite `FreeAndNil(Obj1)` umesto zvanja `Obj1.Free`, a zatim dodeljivanja `nil` objektu `Obj1`. ■

Overloaded (preopterećeni) metodi i konstruktori

Od Delphija 4, Object Pascal podržava overloaded funkcije i metode: možete imati više metoda istog naziva ukoliko imaju različite parametre. Proverom parametara kompajler može odlučiti koju verziju rutine želite da pozovete.

Postoje dva osnovna pravila:

- Iza svake verzije metoda mora slediti ključna reč `overload`.
- Razlike se moraju ogledati u broju ili tipu parametara, ili i broju i tipu. Tip rezultata se ne može koristiti za razlikovanje metoda.

Preopterećenje se može primeniti na globalne funkcije i procedure i na metode klase. Ova mogućnost je naročito relevantna za konstruktore, jer možemo imati više konstruktora i nazvati ih `Create`, što olakšava memorisanje.

NAPOMENA

Istorijski gledano, preopterećenje je dodato jeziku C++ da bi se omogućila upotreba više konstruktora od kojih svaki ima isti naziv (koji odgovara nazivu klase). U Object Pascalu, ova mogućnost je ocenjena kao nepotrebna jer više konstruktora može imati različite specifične nazive. Povećana integracija Delphija sa C++ Builderom je motivisala Borland da ova mogućnost bude dostupna u oba jezika. Kada C++ Builder stvara instancu u klasi Delphi VCL, on traži Delphijev konstruktor naziva Create i ni jedan drugi izuzev Create. Ukoliko Delphi klasa ima konstruktore drugih naziva, oni se ne mogu koristiti iz koda C++ Builder. Zbog toga, kada kreirate klase i komponente koje imate nameru da delite sa programerima C++ Buildera, potrebno je da obratite pažnju i da sve svoje konstruktore nazovete Create i napravite razliku među njima listom parametara (koristeći overload). Ovo ne zahteva Delphi, već C++ Builder da bi koristio Delphi klase. n

Kao primer preopterećenja, ja sam dodao klasi `TDate` dve različite verzije metoda `SetValue`:

```
type
```

```
  TDate = class
  public
    procedure SetValue (y, m, d: Integer); overload;
    procedure SetValue (NewDate: TDateTime); overload;

procedure TDate.SetValue (y, m d: Integer);
begin
  fDate := EncodeDate (y, m, d);
end;

procedure TDate.SetValue(NewDate: TDateTime);
begin
  fDate := NewDate;
end;
```

Posle ovog jednostavnog koraka, ja sam dodao klasi dva različita konstruktora `Create`: jedan bez parametara, koji sakriva unapred određeni konstruktor, a jedan sa inicijalnim vrednostima. Konstruktor bez parametara koristi kao unapred određenu vrednost današnji datum.

```
type
  TDate = class
  public
    constructor Create; overload:
    constructor Create (y, m, d: Integer); overload;

constructor TDate.Create (y, m d: Integer);
begin
  fDate := EncodeDate (y, m, d);
end;

constructor TDate.Create;
begin
  fDate := Date;
end;
```

Postojanje ova dva konstruktora čini mogućim definisanje novog objekta `TDate` na dva različita načina:

```
var
  Dayl, Day2: TDate;
begin
  Dayl := TDate.Create (1999, 12, 25);
  Day2 := TDate.Create; // today
```

Kompletna klasa TDate

U ovom poglavlju sam Vam pokazao delove izvornog koda za različite verzije klase `TDate`. Prva verzija je bila zasnovana na tri celobrojne vrednosti koje čuvaju godinu, mesec i dan; druga verzija koristi polje tipa `TDateTime` koje obezbeđuje Delphi. Evo kompletnog interfejs dela jedinice koja definiše klasu `TDate`:

```
unit Dates;
```

```
interface

type
  TDate = class
  private
    fDate: TDateTime;
    function GetYear: Integer;
  public
    constructor Create; overload;
    constructor Create (y, m, d: Integer); overload;
    procedure SetValue (y, m, d Integer); overload;
    procedure SetValue (NewDate: TDateTime); overload;
    function LeapYear: Boolean;
    procedure Increase (NuniberOfDays: Integer = 1);
    procedure Decrease (NumberOfDays: Integer = 1);
    function GetText: string;
end;

implementation
. . .
```

Cilj novih metoda, `Increase` i `Decrease` (koji imaju unapred određenu vrednost za svoje parametre), je veoma lako shvatiti. Ukoliko ih pozovete bez parametara, oni menjaju datum na prethodni ili naredni dan. Ukoliko je parametar `NumberOfDays` deo poziva, dodaje se ili oduzima taj broj.

```
procedure TDate.Increase (NumberOfDays: Integer = 1);
begin
  fDate := fDate + NueberOfDays;
end;
```

`GetText` kao rezultat daje string sa formatiranim datumom upotrebom funkcije `DateToStr`.

```
function TDate.GetText: string;
begin
  GetText := DateToStr (fDate);
end;
```

U prethodnim sekcijama smo već videli većinu metoda, tako da ja neću dati kompletan listing; možete ga pronaći u primeru ViewDate koji sam napisao da bih testirao klasu. Formular sadrži naslov i šest kontrola koji se mogu upotrebiti da bi se promenio datum. Glavni formular primera ViewDate u vreme izvršavanja možete videti na slici 2.2. Da bi oznaka izgledala lepo, dodelio sam joj veliki font širine formulara, podesio sam svojstvo `Alignment` na `toCenter`, a svojstvo `AutoSize` na `False`.

SLIKA 2.2 *Rezultat primera ViewDate prilikom pokretanja*

Početni kod ovog programa se nalazi u događaju `OnCreate`. U odgovarajućem metodu kreiramo instance klase `TDate`, inicijalizujemo ovaj objekat, a zatim prikazujemo tekstualni opis u svojstvu oznake `Caption`, kao što se može videti na slici 2.2.

```
procedure TDateForm.FormCreate(Sender: TObject);
begin
  TheDay := TDate.Create (1999, 12, 25);
  LabelDate.Caption := TheDay.GetText;
end;
```

`TheDay` je private polje klase formulara `TDateForm`. Usput, naziv klase je Delphi automatski odabrao kada smo promenili svojstvo `Name` u `DateForm`. Objekat je zatim uklonjen uz uklanjanje formulara.

```
procedure TDateForm.FormDestroy(Sender: TObject);
begin
  TheDay.Free;
end;
```

Kada korisnik klikne neku od šest kontrola, potrebno je da primenimo odgovarajuće metode na objekat `TheDay` i da zatim prikažemo novu vrednost datuma u oznaci.

```
procedure TDateForm.BtnTodayClick(Sender: TObject);
begin
  TheDay.SetValue (Date);
  LabelDate.Caption := TheDay.GetText;
end;
```

Alternativni način na koji možete napisati poslednji metod je da uklonite aktuelni objekat i kreirate novi.

```
procedure TDateForm.BtnTodayClick(Sender: TDbject);
var
  NewDay: TDate;
begin
  TheDay.Free;
  NewDay := TDate.Create;
  TheDay := NewDay;
  LabelDate.Caption := TheDay.GetText;
end;
```

Pri ovim uslovima, ovo nije naročito dobar pristup (jer kreiranje novog objekta i uklanjanje postojećeg zahteva dosta vremena, a potrebno je samo da promenimo vrednost postojećeg objekta), ali mi je omogućio da Vam pokažem nekoliko tehnika Object Pascala. Prva stvar koju treba primetiti je da uklanjamo prethodni objekat pre nego što dodelimo novi. Ova operacija dodeljivanja, zapravo, zamenjuje referencu, ostavljajući objekat u memoriji (čak i ako se nijedan pokazivač ne odnosi na nju). Kada jedan objekat dodeljujete drugom objektu, Delphi jednostavno kopira referencu objekta u memoriji u novi objekat/referencu.

Ukoliko zaista želite da promenite podatak unutar postojećeg objekta, kopirajte svako polje, ili obezbedite specifičan metod za kopiranje internih podataka. Neke klase VCL sadrže metod `Assign` koji obavlja "dubinsko" kopiranje (deep copying). Da bih bio precizniji, sve klase VCL koje nasleđuju od `TPersistent` sadrže metod `Assign`, ali većina nasleđuje od `TComponent` koja taj metod ne implementira, pozivajući se na izuzetak prilikom poziva.

Nasleđivanje od postojećih tipova

Često imamo potrebu da koristimo malo drugačiju verziju postojeće klase koju smo napisali ili koju nam je neko dao. Na primer, možda je potrebno da dodate novi metod ili da malo promenite postojeći. To lako možete učiniti izmenom originalnog koda, izuzev ukoliko želite da imate mogućnost korišćenja dve različite verzije klase u različitim situacijama. Takođe, ukoliko je klase originalno napisao neko drugi (uključujući Borland), možda želite da Vaše izmene čuvate odvojeno.

Tipična alternativa je da načinite kopiju originalne definicije tipa, promenite kod da biste podržali nove mogućnosti, i dodelite novi naziv rezultujućoj klasi. Ovo će, možda, funkcionisati, ali, takođe, može stvoriti probleme: dupliranjem koda, takođe, duplirate i greške; ukoliko želite da dodate novu funkciju, potrebno je da je dodate dva ili više puta, već prema broju kopija originalnog koda koje ste načinili. Ovaj pristup kao rezultat daje dva potpuno različita tipa podataka, tako da Vam kompajler ne može pomoći da iskoristite prednost sličnosti između dva tipa podataka.

Da biste rešili ovakve tipove problema u izražavanju sličnosti između klasa, Object Pascal Vam omogućava da definišete novu klasu direktno iz postojeće. Ova tehnika je poznata kao *nasleđivanje* (inheritance, ili subclassing, ili derivation) i čini jedan od fundamentalnih elemenata obejktno orijentisanih programskih jezika. Da biste nasledili od postojeće klase, potrebno je da samo naznačite klasu na početku deklaracije potklase. Na primer, Delphi to čini automatski svaki put kada kreirate novi formular:

```
type
  TForm1 = class(TForm)
  end;
```

Ova jednostavna definicija označava da klasa `TForm1` nasleđuje sve metode, polja, svojstva i događaje od klase `TForm`. Možete upotrebiti bilo koji javni metod klase `TForm` nad objektom tipa `Tform1`. `TForm` će odmah naslediti neke od svojih metoda od neke druge klase i tako dajle, sve do klase `TObject`.

Kao jednostavan primer nasleđivanja možemo malo izmeniti program ViewDate, izvodeći ga iz klase `TDate` i menjajući jednu od funkcija, funkciju `GetText`. Ovaj kod možete pronaći u fajlu `DATES.PAS` primera ViewD2.

```
type
  TNewDate = class (TDate)
  public
    function GetText: string;
  end;
```

U ovom primeru, `TNewDate` je izveden iz `TDate`. Uobičajeno je reći da je `TDate` predak (ancestor) ili roditelj (parent) klase `TNewDate` i da je `TNewDate` potklasa (subclass), klasa-naslednik (descendant) ili dete-klasa (child) klase `TDate`.

Da bih implementirao novu verziju funkcije `GetText`, ja sam upotrebio funkciju `FormatDateTime`, koja koristi (između ostalog) unapred određene nazive meseci koji su dostupni pod Windowsom; ovi nazivi zavise od korisničkog podešavanja regiona i jezika. Mnoge od ovih vrednosti Delphi, zapravo, kopira u konstante definisane u biblioteci, kao što su `LongMonthNames`, `ShortMonthNames` i mnoge druge koje možete pronaći pod temom *Currency and dat/time formatting variables* u Delphi Help fajlu. Evo metoda `GetText`, gde *'dddddd'* označava dugački format podatka:

```
function TNewDate.GetText: string;
begin
  GetText := FormatDateTime ('dddddd', fDate);
end;
```

SAVET

Korišćenjem regionalnih informacija, program ViewD2 se automatski adaptira prema različitim korisničkim podešavanjima Windowsa. Ukoliko pokrenete isti program na kompjuteru na kojem se koriste regionalna podešavanja prema jeziku koji nije engleski, program će automatski prikazati nazive meseci na tom jeziku. Da biste testirali ponašanje, potrebno je da samo promenite regionalna podešavanja; nije Vam potrebna nova verzija Windowsa. Primetićete da se promene regionalnih podešavanja odmah reflektuju na programe koji se izvršavaju. ■

Kada smo jednom definisali novu klasu, potrebno je da upotrebimo novi tip podataka u kodu formulara primera ViewD2. Jednostavno definišite tip `TheDay` objekta kao `TNewDate` i pozovite njegov konstruktor iz metoda `FormCreate`.

```
type
  TDateForm = class(TForm)
    . . .
  private
    TheDay: TNewOate; // updated declaration
  end;

procedure TDateForm.FormCreate(Sender: TObject);
begin
  TheDay := TNewDate.Create (1998, 12, 25); // updated
  DateLabel.Caption := TheDay.GetText;
end;
```

Primer ViewD2 će se pravilno izvršavati bez ikakvih izmena. Klasa `TNewDate` nasleđuje svoje metode da bi uvećala datum, dodala određeni broj dana i tako dalje. Uz to, stariji kod, koji poziva ove metode, će i dalje funkcionisati. Zapravo, da bismo pozvali novi metod `GetText`, ne moramo promeniti izvorni kod! Delphi kompajler će automatski povezati taj poziv sa novim

metodom. Izvorni kod svih ostalih događaja ostaje isti, mada mu se značenje bitno menja, kao što to novi izlaz pokazuje (videti sliku 2.3).

SLIKA 2.3 *Izlaz programa ViewD2 sa nazivima meseca i dana u nedelji koji zavise od regionalnog podešavanja Windowsa*

Zaštićena polja (protected) i enkapsulacija

Kod metoda `GetText` klase `TNewDate` se kompajlira samo ukoliko je napisan u okviru jedinice klase `TDate`. Zapravo, on pristupa privatnom polju `fDate` klase pretka. Ukoliko želimo da unesemo klasu naslednika u novu jedinicu, moramo polje `fDate` deklarisati kao zaštićeno (protected) ili moramo dodati mali, po mogućstvu zaštićeni, metod u klasu pretka koja će pročitati vrednost privatnog polja.

Mnogi programeri veruju da je prvo rešenje uvek najbolje, jer će deklarisanje većine polja zaštićenim učiniti da će klasa moći lakše da se proširi i da će olakšati pisanje potklasa. Ipak, ovim se narušava ideja enkapsulacije. U velikoj hijerarhiji klasa, promena definicije nekih zaštićenih polja osnovne klase postaje teška koliko i promena globalne strukture podataka. Ukoliko deset izvedenih klasa prethodi ovoj klasi, promena njene definicije znači moguću izmenu koda u svakoj od deset klasa.

Drugim rečima, proširenje i enkapsulacija često postaju konfliktni ciljevi. Kada se to desi, trebalo bi da favorizujete enkapsulaciju. Ukoliko to možete učiniti, a da ne žrtvujete fleksibilnost, onda će to biti još bolje. Često se ovo prelazno rešenje može postići upotrebom virtuelnih metoda, teme koju ću razmatrati u odeljku "Kasno povezivanje i polimorfizam". Ukoliko odlučite da ne koristite enkapsulaciju da biste dobili brže kodiranje potklasa, tada Vaš dizajn možda neće slediti objektno orijentisane principe.

PRISTUPANJE ZAŠTIĆENIM PODACIMA DRUGIH KLASA

Videli smo da su u Delphiju podaci klasa `private` i `protected` dostupni bilo kojoj funkciji ili metodu koji se javlja unutar iste jedinice klase. Na primer, razmislite o ovoj jednostavnoj klasi (deo primera Protection):

```
type
  TTest = class
  protected
    ProtectedData: Integer;
  public
    PublicData: Integer;
    function GetValue: string;
  end;
```

Metod `GetValue` jednostavno kao rezultat daje string sa dve celobrojne vrednosti:

```
function TTest.GetValue: string;
begin
  Result := Format ('Public: %d, Protected: %d',
    [PublicData, ProtectedData]);
end;
```

Kada jednom smestite ovu klasu u zasebnu jedinicu, nećete moći da direktno pristupite njenom zaštićenom delu iz drugih jedinica. Slično, ukoliko napišete sledeći kod:

```
procedure TForml.ButtonlClick(Sender: TObject);
var
  Obj: TTest;
begin
    Obj := TTest.Create;
    Obj.PublicData := 10;
    Obj.ProtectedOata := 20; // won't compile
    ShowMessage (Obj.GetValue);
    Obj Free;
end;
```

kompajler će prijaviti grešku *Undeclared identifier: "ProtectedData"* (Nedeklarisana oznaka: "Zaštićeni podaci"). U ovom trenutku ćete, možda, pomisliti da ne postoji način da pristupite zaštićenim podacima klase definisane u drugoj jedinici. (To je ono što uputstva za Delphi i većina knjiga o Delphiju kažu.) Ipak, postoji zaobilaznica. Razmislite šta se dešava kada kreirate naizgled beskorisnu izvedenu klasu kao što je:

```
type
  TFake class (TTest);
```

Sada, ukoliko načinite direktno obraćanje obejkta novoj klasi i pristupite zaštićenim podacima preko nje, evo kako će kod izgledati:

```
procedure TForml.Button2CliCk(Sender: TObject);
var
  Obj: TTest;
begin
  Obj TTest.Create;
  Obj.PublicData := 10;
  TFake (Obj).ProtectedData := 20; // compiles!
  ShowMessage (Obj.GetValue);
  Obj Free;
end;
```

Ovaj kod se kompajlira i izvršava korektno, kao što možete videti izvršavanjem programa Protected. Kako je moguće da ovakav pristup funkcioniše? Ukoliko razmislite, klasa `TFake` automatski nasleđuje zaštićena polja osnovne klase `TTest`, i pošto se klasa `TFake` nalazi u okviru iste jedinice kao i kod koji pokušava da pristupi podacima u nasleđenim poljima, zaštićeni podaci su dostupni. Kao što očekujete, ukoliko premestite deklaraciju klase `TFake` u drugu jedinicu, program više neće biti moguće kompajlirati.

Sada kada sam Vam pokazao kako da to učinite, moram Vas upozoriti da narušavanjem mehanizma zaštite klasa na ovaj način lako dolazi do grešaka u Vašem programu (pristupanjem podacima kojima zapravo ne biste smeli da pristupate), i to je suprotno od dobrobiti OOP tehnike. Ipak, postoje situacije kada je upotreba ove tehnike najbolje rešenje, kao što ćete videti u izvornom kodu VCL i kodu mnogih Delphi komponenata. Dva jednostavna primera, koja odmah padaju na um, su pristupanje svojstvu `Text` klase `TControl` i pozicijama `Row` i `Col` kontrole. Ove dve ideje su demonstrirane primerima TextProp i `DBGridCol` respektivno. (Ovi primeri su prilično napredni, te preporučujem da samo programeri sa dobrim iskustvom u Delphiju u ovom trenutku pročitaju primere — ostali se na njih mogu kasnije vratiti.) Mada prvi primer predstavlja razuman primer upotrebe kategorizovanog krakera (cracker), primer DBGrid za `Row` i `Col` je, zapravo, primer brojanja, onaj koji ilustruje rizik pristupanju delova klase koje je kreator klase odlučio da ne prikaže. Red i kolona `DBGrida` nemaju isto značenje kao kod `DrawGrid` i `StringGrid` (osnovnih klasa). Prvo, `DBGrid` ne broji fiskirane ćelije kao aktuelne ćelije (time se ćelije razlikuju od dekoracije), tako da indeksi redova i kolona moraju da budu prilagođeni prema bilo kojoj dekoraciji koja je aktivna za tabelu (a i one se veoma brzo mogu promeniti). Drugo, `DBGrid` predstavlja virtuelni pogled na podatke tabele. Kada pomerate pogled u `DBGridu`, podaci se mogu pomerati, ali trenutno selektovani red se ne mora promeniti.

Ova tehnika se često opisuje kao rez (hack), i treba je izbegavati kada god je to moguće. Problem ne predstavlja pristupanje zaštićenim podacima klase iz iste jedinice već deklarisanje klase sa jedinom svrhom pristupanja zaštićenim podacima postojećeg objekta druge klase! Loša strana ove tehnike je u kodiranju objekta jedne klase iz druge klase.

Nasleđivanje i kompatibilnost tipova podataka

Pascal je striktno jezik tipova podataka. To znači da ne možete, na primer, da dodelite celobrojnu vrednost promenljivoj tipa `Boolean`, bar ne bez eksplicitnog pretvaranja. Pravilo je da su dve vrednosti kompatibilne po tipu samo ukoliko su istog tipa podataka, ili (da budemo precizniji) ukoliko njihovi tipovi podataka imaju isti naziv i njihova definicija potiče iz iste jedinice.

Postoji važan izuzetak od ovog pravila kada su u pitanju tipovi klasa. Ukoliko deklarišete klasu, recimo `TAnimal`, i iz nje izvedete novu klasu, recimo `TDog`, možete posle toga dodeliti objekat tipa `TDog` promenljivoj tipa `TAnimal`. Ovo je moguće jer je pas životinja! Dakle, mada Vas ovo može iznenaditi, oba naredna poziva konstruktora su ispravna:

```
var
  MyAnimal1, MyAnimal2: TAnimal;
begin
  MyAnimal1 := TAnimal.Create;
  MyAnimal2 := TDog.Create;
```

Kao opšte pravilo, možete koristiti objekat klase naslednika svaki put kada se očekuje objekat klase pretka. Ipak, obrnuto nije dozvoljeno; ne možete koristiti objekat klase pretka kada se očekuje objekat klase naslednika. Da bismo pojednostavili objašenjenje, evo objašnjenja u terminima koda:

```
MyAnimal := MyDog; // This is OK
MyDog := MyAnimal; // This is an error!!!
```

Pre nego što se pozabavimo implikacijama ove važne mogućnosti jezika, mogli biste da isprobate primer Animals1, koji definiše dve jednostavne klase `TAnimal` i `TDog`:

```
type
  TAnimal = class
  public
    constructor Create;
    function GetKind: string;
  private
    Kind: string;
  end;

  TDog = class (TAnimal)
  public
    constructor Create;
  end;
```

Dva metoda `Create` jednostavno određuju vrednost za kontrolu `Kind`, koja se dobija funkcijom `GetKind`. Formular koji prikazuje ovaj primer, prikazan na slici 2.4, sadrži privatno polje tipa `TAnimal`. Instanca ove klase se kreira i inicijalizuje kada se kreira formular i svaki put kada se odabere jedna od kontrola (radio buttons).

```
procedure TFormAnimals.FormCreate(Sender: TObject);
begin
  MyAnimal := TAnimal.Create;
end;

procedure TFormAnimals.RbtnDogClick(Sender: TObject);
begin
  MyAnimal.Free;
  MyAnimal :=  TDog.Create;
end;
```

SLIKA 2.4 *Formular primera Animals1*

Konačno, kontrola Kind poziva metod `GetKind` za trenutno odabranu životinju i prikazuje rezultat u oznaci:

```
procedure TFormAnimals.BtnKindClick(Sender: TObject);
begin
  KindLabel.Caption := MyAnirnal .GetKind;
end;
```

Kasno povezivanje i polimorfizam

Pascalove funkcije i procedure se obično zasnivaju na statičkom povezivanju (static binding), koje se još naziva i ranim povezivanjem (early binding). To znači da poziv metodu rešava kompajler ili linker koji zamenjuje zahtev pozivom određene memorijske adrese na kojoj se funkcija ili procedura nalazi. (Ovo je poznato kao adresa (address) funkcije.) Objektno orijentisani programski jezici omogućavaju upotrebu još jednog oblika povezivanja, poznatog kao dinamičko povezivanje (dynamic binding), ili kasno povezivanje (late binding). U ovom slučaju, stvarna adresa metoda koji treba pozvati se određuje u vreme izvršavanja na osnovu tipa instance koja je upotrebljena za poziv.

Prednost ove tehnike je poznata kao polimorfizam (polymorphism). Polimorfizam znači da možete da napišete poziv metodu, dodeljujući ga promenljivoj, ali koji će metod zapravo Delphi pozvati, zavisi od tipa objekta na koji se odnosi promenljiva. Delphi ne može odrediti, sve do vremena izvršavanja, pravu klasu objekta na koji se odnosi promenljiva, prosto zbog pravila komaptibilnosti tipova koje smo razmatrali u prethodnom odeljku.

NAPOMENA

Termin polimorfizam je glomazan. Pogled u rečnik nam govori da se, uopšte uzev, odnosi na nešto što postoji u više od jednog oblika. U smislu OOP-a odnosi se na činjenicu da možda postoji nekoliko verzija datog metoda i da se jedan poziv metodu može odnositi na bilo koju od verzija. ■

Na primer, pretpostavimo da klasa i njene potklase (recimo `TAnimal` i `TDog`) definišu isti metod, i da taj metod ima kasno povezivanje. Sada taj metod možete primeniti na generičku promenljivu, kao što je `MyAnimal`, koja se u vreme izvršavanja može odnositi bilo na objekat klase `TAnimal` bilo na objekat klase `TDog`. Stvarni metod koji treba pozvati se određuje u vreme izvršavanja, već prema klasi aktuelnog objekta.

Primer Animals2 unapređuje program Animals1 da bi demonstrirao ovu tehniku. U novoj verziji, klase `TAnimal` i `TDog` sadrže novi metod `Voice`, koji treba da proizvede zvuk koji odabrana životinja proizvodi, i kao tekst i kao zvuk. Ovaj metod je definisan kao `virtual` u klasi `TAnimal`, a kasnije se zaobilazi kada definišemo klasu `TDog`, upotrebom reči `virtual` i `override`:

```
type
  TAnimal = class
  public
    function Voice: string; virtual;

  TDog = class (TAnimal)
  public
    function Voice: string; override;
```

Naravno, oba metoda je potrebno implementirati. Evo jednostavnog pristupa:

```
uses
  MMSystem:

function TAnimal.Voice: string;
begin
  Voice := 'Voice of the animal';
  PlaySound ('Anirn.wav',  0, snd_Async);
end;

function TDog.Voice: string;
begin
  Voice := 'Arf Arf';
  PlaySound ('dog.wav',  0, snd_Async);
end;
```

SAVET

U ovom primeru je korišćen poziv API funkciji `PlaySound`, definisanoj u jedinici MMSystem. Prvi parametar ove funkcije je naziv WAV fajla ili sistemskog fajla koji želite da upotrebite. Drugi parametar se odnosi na opcioni resurs fajl koji sadrži zvuk. Treći parametar označava (između ostalih opcija) da li poziv treba da bude sinhroni ili asinhroni, to jest, da li program treba da sačeka da se zvuk završi pre nego što nastavi izvršavanje narednih linija koda. ■

Šta je efekat poziva `MyAnimal.Voice`? Zavisi. Ukoliko se promenljiva `MyAnimal` trenutno odnosi na objekat klase `TAnimal`, biće pozvan metod `TAnimal.Voice`. Ukoliko se odnosi na klasu `TDog`, biće pozvan metod `TDog.Voice`. Ovo se dešava samo zato što je funkcija virtuelna (`virtual`).

Poziv `MyAnimal.Voice` će funkcionisati za objekat koji je instanca bilo koje klase naslednika klase `TAnimal`, pa čak i klase koja je definisana posle poziva metoda ili je van delokruga. Kompajleru nije potrebno da zna o svim klasama naslednicima da bi poziv učinio kompatibilnim; potrebna je samo klasa predak. Drugim rečima, poziv `MyAnimal.Voice` je kompatibilan sa svim budućim potklasama `TAnimal`.

Ovo je ključni tehnički razlog zašto objektno orijentisani programski jezici favorizuju fleksibilnost. Možete napisati kod koji koristi klase u okviru hijerarhije, a da ne morate da poznajete određene klase koje su deo te hijerarhije. Drugim rečima, hijerarhija — i program — se još uvek mogu proširiti, čak i kada ste napisali hiljade linija koda koje ga koriste. Naravno, postoji jedan uslov — potrebno je da klase koje su naslednici hijerarhije budu pažljivo dizajnirane.

Program Animals2 demonstrira upotrebu ovih novih klasa i sadrži formular sličan prethodnom primeru. Naredni kod se izvršava kada kliknete kontrolu:

```
procedure TFormAnirnals.BtnVerseClick(Sender: TObject);
begin
  LabelVoice.Caption := MyAnimal.Voice;
end;
```

Na slici 2.5 možete videti primer izlaza ovog programa. Njegovim izvršavanjem ćete, takođe, čuti odgovarajuće zvuke koje proizvodi API funkcija `PlaySound`.

SLIKA 2.5 *Izlaz primera Animals2*

Prevazilaženje, ponovno definisanje i ponovno uvođenje metoda

Kao što smo upravo videli, da biste zaobišli kasno povezivanje metoda u klasi nasledniku, potrebno je da upotrebite ključnu reč `override`. Imajte na umu da je ovo moguće samo ukoliko je metod u klasi pretku definisan kao `virtual`. Inače, ukoliko je to statički metod, ne bi postojao način za aktiviranje kasnog povezivanja, a da se ne promeni kod u klasi pretku.

Pravila su jednostavna. Metod koji je definisan kao statički, ostaje statički u svakoj potklasi izuzev ukoliko ga ne sakrijete novim virtuelnim metodom koji ima isti naziv. Metod definisan kao virtuelni (`virtual`) se može kasno povezivati u svakoj od potklasa. Ne postoji način da se ovo promeni zbog načina na koji kompajler generiše kod za metode koji se kasno povezuju.

Da biste ponovo definisali statički metod, jednostavno dodajte metod potklasi tako da sadrži neke parametre ili različite parametre od originalnog metoda, bez daljih specifikacija. Da biste zaobišli `virtual` metod, morate navesti iste parametre i upotrebiti ključnu reč `override`:

```
type
  MyClass = class
    procedure One; virtual;
    procedure Two; (static method)
  end;

  MySubCass = class (MyClass)
    procedure One; override;
    procedure Two;
  end;
```

Postoje dva tipična načina za zaobilaženje metoda. Jedan je da zamenite metod klase pretka novom verzijom. Drugi način je da dodate neki kod više postojećem metodu. To se može postići upotrebom ključne reči `inherited` pri pozivu nekog metoda klase prethodnika. Na primer, možete napisati

```
procedure MySubClass.One;
begin
  // new code
  . . .
  // call inherited procedure MyClass One
  inherited One;
end;
```

Možda se pitate zašto morate da koristite ključnu reč `override`. U drugim jezicima, kada ponovo definišete metod u potklasi, Vi automatski zaobilazite originalni. Ipak, postojanje specifične ključne reči omogućava kompajleru da proveri veze između naziva metoda klase pretka i potklase (pogrešno napisan naziv ponovno definisane funkcije je česta greška u drugim OOP jezicima), proveri da li je metod virtuelni u klasi pretku, i tako dalje.

Dalje, ukoliko definišete statički metod u bilo kojoj klasi nasledniku klase biblioteke, neće biti problema, iako se bibilioteka ažurira novim virtuelnim metodom koji ima isti naziv kao i metod koji ste Vi definisali. Kako Vaš metod nije označen ključnom reči `override`, biće smatran za zasebni metod, a ne za novu verziju metoda koji je dodat biblioteci (nešto što bi verovatno narušilo Vaš kod).

Podrška preopterećenju predstavljena u Delphiju 4 je doprinela većoj složenosti. Potklase mogu obezbediti novu verziju metoda upotrebom ključne reči `override`. Ukoliko metod sadrži drugačije parametre nego u verziji osnovne klase, on u osnovi postaje preopterećeni metod; inače će zameniti metod osnovne klase. Evo primera:

```
type
  TMyC1ass = class
    procedure One;
  end;

  TMySubC1ass = class (TMyClass)
    procedure One (S: string); overload;
  end;
```

Primetite da metod nije potrebno označiti kao overrload u osnovnoj klasi. Ipak, ukoliko je metod u osnovnoj klasi virtuelni, kompajler prijavljuje grešku *Method 'One' hides virtual method of base type 'TMyClass'* (Metod 'Jedan' sakriva virtuelni metod osnovnog tipa 'TMyClass'). Da biste izbegli ovakvu poruku kompajlera i da biste kompajleru preciznije nagovestili svoje namere, možete upotrebiti novu direktivu `reintroduce`:

```
type
  TMyClass = class
    procedure One; virtual;
  end;

  TMySubClass = class (TMyC1ass)
    procedure One (S: string); reintroduce; overload;
  end;
```

Ovaj kod možete da pronađete u primeru Reintr i da dalje njime eksperimentišete.

Virtuelni nasuprot dinamičkih metoda

U Delphiju postoje dva načina na koje možete aktivirati kasno povezivanje. Metod možete deklarisati kao virtuelni (`virtual`), kao što smo to ranije videli, ili ga deklarisati kao dinamički (`dynamic`). Sintaksa ove dve ključne reči je identična, a i rezultat njihove upotrebe je identičan. Ono što se razlikuje je interni mehanizam koji koristi kompajler da primeni kasno povezivanje.

Virtuelni metodi se zasnivaju na *tabeli virtuelnih metoda* (VMT, takođe poznatoj kao *vtable*). Tabela virtuelnih metoda je niz adresa metoda. Za poziv virtelnom metodu kompajler generiše kod kojim se skače na adresu koja se čuva u n-tom slotu tabele virtuelnog metaobjekta.

Tabele virtuelnih metoda omogućavaju brzo izvršavanje poziva metodima. Njihov glavni nedostatak je taj da zahtevaju element za svaki virtuelni metod za svaku klasu naslednika, iako se metod ne zaobilazi u potklasama. Ponekad se ponavljaju elementi tabele virtuelnih metoda kroz hijerarhiju klasa (čak i za metode koji nisu ponovo definisani). To može da zahteva dosta memorije samo za čuvanje iste adrese metoda veći broj puta.

Dinamički pozivi metodima, s druge strane, se obavljaju upotrebom jedinstvenog broja koji identifikuje metod. Potraga za odgovarajućom funkcijom je sporija od jednostavnog pretraživanja virtuelnih metoda. Prednost je u tome da se elementi dinamičkih metoda ponavljaju samo u naslednicima ukoliko naslednici zaobilaze metod. Za veliku hijerarhiju, upotreba dinamičkih umesto virtuelnih metoda može značajno uštedeti memoriju uz minimalan gubitak prilikom izvršavanja.

Iz perspektive programera, razlika između ova dva pristupa leži samo u različitim internim reprezentacijama i maloj razlici u upotrebi memorije i brzini izvršavanja. Izuzumajući ovo, metodi `virtual` i `dynamic` su identični.

Rukovanje porukama

Metod sa kasnim povezivanjem se može upotrebiti za rukovanje porukama Windowsa, mada se tehnika ponešto razlikuje. Za ovu svrhu Delphi obezbeđuje još jednu direktivu, `message`, za definisanje metoda za rukovanje porukama koje moraju da budu procedure sa jednim `var` parametrom. Direktivu `message` sledi broj Windowsove poruke kojom rukuje metod. Na primer, naredni kod Vam omogućava da rukujete korisnički definisanom porukom, označenom numeričkom vrednošću Windowsove konstante `wm_User`:

```
type
  TForm1 = class (TForm)
    ...
    procedure WmUser (var Msg: TMessage);
      message wm_User;
  end;
```

Naziv procedure i stvarni tip parametara zavise od Vas, mada postoji veliki broj unapred definisanih tipova slogova za različite Windowsove poruke. Ova tehnika može da bude izuzetno korisna za iskusne programere Windowsa koji znaju sve o Windowsovim porukama i API funkcijama.

NAPOMENA

Mogućnost rukovanja Windowsovim porukama i pozivima API funkcija kao kada programirate Windows upotrebom C jezika, može zastrašiti neke programere a druge oduševiti. Ali u Delphiju, kada pišete Windows aplikacije, često ćete imati potrebu da koristite `message` metode. Potrebu da se bavite porukama niskog nivoa i API funkcijama ćete imati jedino kada pišete složene komponente u Delphiju. ■

Apstraktni metodi

Ključna reč `abstract` se koristi za označavanje metoda koji će biti definisani samo u potklasama aktuelne klase. Direktiva `abstract` u potpunosti definiše metod; to nije deklaracija koja prosleđuje. Ukoliko pokušate da obezbedite definiciju za metod, kompajler će se buniti. U Object Pascalu možete kreirati instance klasa koje sadrže apstraktne metode. Ipak, kada to pokušate da učinite, Delphijev 32-bitni kompajler će dati upozorenje: *Constructing instance of <class name> abstract methods* (Konstruisanje instance <naziv klase> sadrži apstraktne metode). Ukoliko se desi poziv apstraktnog metoda u vreme izvršavanja, Delphi će se pozvati na izuzetak, kao što je to pokazano u narednom primeru Animals3.

NAPOMENA

C++ i Java upotrebljavaju striktniji pristup: u ovim jezicima ne možete kreirati instance apstraktnih klasa. ■

Možda se pitate zašto biste koristili apstraktne metode. Razlog leži u upotrebi polimorfizama. Kada bi klasa `TAnimal` sadržala apstraktni metod `Voice`, svaka potklasa bi mogla da ga ponovo definiše. Prednost je u tome što sada možete koristiti generički objekat `MyAnimal` da biste se referisali na svaku životinju definisanu potklasom i pozvali ovaj metod. Kada ovaj metod ne bi bio prisutan u nasleđivanju klase `TAnimal`, kompajler ne bi dopustio poziv koji izvodi statički tip provere. Upotrebom generičkog objekta `MyAnimal` možete pozvati samo metod definisan sopstvenom klasom, `TAnimal`.

Ne možete pozivati metode koje obezbeđuju potklase, izuzev ukoliko klasa roditelj ne sadrži najmanje jednu deklaraciju ovog metoda — u formi apstraktnog metoda. Sledeći primer, Animals3, demonstrira upotrebu apstraktnih metoda i greške prilikom apstraktnog poziva. Evo interfejsa klasa ovog novog primera:

```
type
  TAnimal = class
  public
    constructor Create;
    function GetKind: string;
    function Voice: string; virtual; abstract;
  private
    Kind: string;
  end;

  TDog = class (TAnimal)
  public
    constructor Create;
    function Voice: string; override;
    function Eat: string; virtual;
  end;

  TCat = class (TAnimal)
  public
    constructor Create;
    function Voice: string; override;
    function Eat: string; virtual;
  end;
```

Najinteresantniji deo je definicija klase `TAnimal`, koja sadrži virtuelni apstraktni metod: `Voice`. Takođe je važno primetiti da svaka izvedena klasa zaobilazi ovu definiciju i dodaje novi virtuelni metod, `Eat`. Koje su implikacije ova dva različita pristupa? Da biste pozvali funkciju `Voice`, možete jednostavno napisati isti kod kao u prethodnoj verziji programa:

```
LabelVoice.Caption := MyAnimal.Voice;
```

Kako možemo pozvati metod `Eat`? Ne možemo ga primeniti na objekat klase `TAnimal`. Iskaz

```
LabelVoice.Caption := MyAnimal.Eat;
```

generiše grešku prilikom kompajliranja *Field identifier expected* (Očekuje se identifikator polja).

Da biste rešili ovaj problem, možete upotrebiti tip inforamcije prilikom izvršavanja (RTTI) da biste dodelili objekat `TAnimal` objektu `TDog` ili `TCat`; međutim, bez ispravnog prevođenja program će se pozvati na izuzetak. Primer ovakvog pristupa ćete videti u narednom odeljku. Dodavanje definicije metoda klasi `TAnimal` je tipično rešenje problema, i prisutstvo ključne reči `abstract` favorizuje ovaj izbor.

Tip informacije prilikom izvršavanja

Pravila kompatibilnosti Object Pascala za klase naslednike Vam omogućavaju da koristite klase naslednike tamo gde se očekuje klasa predak. Kao što sam ranije naglasio, obrnuto nije moguće.

Pretpostavimo sada da klasa `TDog` sadrži metod `Eat`, koji ne postoji u klasi `TAnimal`. Ukoliko se promenljiva `MyAnimal` odnosi na psa, trebalo bi da je moguće pozvati ovu funkciju. Međutim, ukoliko pokušate, a promenljiva se odnosi na neku drugu klasu, rezultat je greška. Eksplicitnom kategorizacijom možemo proizvesti veliku grešku prilikom izvršavanja (ili što je još gore, suptilni problem sa prepisivanjem memorije), jer kompajler ne može da odredi da li je tip objekta korektan i da li metodi koje pozivamo stvarno postoje.

Da bismo rešili problem, možemo upotrebiti tehnike koje se zasnivaju na tipu informacije prilikom izvršavanja. U osnovi, kako svaki objekat "zna" svoj tip i roditeljske klase, možemo zatražiti ovu informaciju operatorom `is` ili upotrebom nekih metoda klase `TObject` (koju ćemo razmatrati u narednom poglavlju). Parametri operatora `is` su objekat i tip klase, a rezultujuća vrednost je Boolean:

```
if MyAnimal is TDog then
  ...
```

Izraz `is` daje kao vrednost `True` samo ukoliko se objekat `MyAnimal` trenutno odnosi na objekat klase TDog ili je tipa izvedenog iz klase `TDog`. To znači, da ukoliko testirate da li je objekat `TDog` tipa `TAnimal`, test će uspeti. Drugim rečima, izraz daje kao rezultat vrednost `True` ukoliko možete bezbedno dodeliti objekat (`MyAnimal`) promenljivoj tipa podataka (`TDog`).

Sada kada sigurno znate da je životinja pas, možete bezbedno izvršiti kategorizaciju (ili konverziju tipova). Ovo direktno dodeljivanje možete postići ukoliko napišete sledeći kod:

```
if MyAnimal is TDog then
begin
  MyDog := TDog (MyAnimal);
  Text :=MyDog.Eat;
end;
```

Ista operacija se direktno može postići drugim RTTI operatorom, `as`, koji konvertuje objekat samo ukoliko je zahtevana klasa kompatibilna sa aktuelnom klasom. Parametri operatora `as` su objekat i tip klase, a rezultat je objekat konvertovan u novi tip klase. Možemo napisati sledeći isečak:

```
MyDog := MyAnimal as TDog;
Text := MyDog.Eat;
```

Ukoliko želimo da samo pozovemo funkciju `Eat`, možemo upotrebiti još kraću notaciju:

```
(MyAnimal as TDog).Eat;
```

Rezultat ovog izraza je objekat tipa podataka klase `TDog`, tako da možete na njega primeniti bilo koji metod te klase. Razlika između tradicionalnog pristupa i upotrebe operatora `as` je da se drugi pristup poziva na izuzetak ukoliko tip objekta nije kompatibilan sa tipom koji pokušavate da upotrebite. Izuzetak koji se poziva je `EInvalidCast` (izuzetke ćemo opisati na kraju ovog poglavlja).

Da biste izbegli ovaj izuzetak, upotrebite operator `is` i, ukoliko je rezultat `True`, načinite jednostavnu konverziju (zapravo, ne postoji razlog da upotrebljavate operatore `is` i `as` jedan za drugim i dva puta proveravate tip).

```
if  MyAnimal is  TDog then
  TDog(MyAnimal).Eat;
```

Oba operatora RTTI su veoma korisna u Delphiju jer ćete često imati potrebu da napišete generički kod koji se može upotrebiti sa velikim brojem komponenata istog tipa ili čak različitih tipova. Kada se komponenta prosleđuje kao parametar metodu koji reaguje na događaj, koristi se generički tip podataka (`TObject`), tako da ćete često morati da ga dodelite originalnom tipu komponente:

```
procedure TForm1.ButtonClick (Sender: TObject);
begin
  if Sender is TButton then
    ...
end;
```

Ovo je uobičajena tehnika u Delphiju i koristiću je u velikom broju primera koje ćete naći u ovoj knjizi. U Poglavlju 4 ponovo ćemo razmatrati operatore `is` i `as` dok ćemo obratiti pažnju na neke alternativne tehnike RTTI koje se zansivaju na metodima klase `TObject`. Dva operatora RTTI, `is` i `as`, su neverovatno moćna i možda ćete doći u iskušenje da ih posmatrate kao standardne programske konstrukte. Mada su zaista moćni, verovatno bi trebalo da ograničite njihovu upotrebu na specijalne slučajeve. Kada je potrebno da rešite složen problem koji uključuje nekoliko klasa, prvo pokušajte da upotrebite polimorfizam. Samo u specijalnim slučajevima, kada se samo polimorfizam ne može upotrebiti, trebalo bi da upotrebite operatore RTTI. *Nemojte koristiti RTTI umesto polimorfizma.* To je loša programerska praksa i može proizvesti sporije programe. RTTI, zapravo, ima negativan uticaj na performanse jer mora da prođe kroz hijerarhiju klasa da bi se proverilo da li je konverzija korektna. Kao što smo videli, virtuelni metodi zahtevaju samo pretraživanje memorije, što je mnogo brže.

Vizuelno nasleđivanje formulara

Da biste bolje shvatili izvođenje između klasa, možete upotrebiti vizuelno nasleđivanje formulara. Ukratko, možete jednostavno naslediti formular od postojećeg formulara, dodajući nove komponente ili menjajući svojstva postojećih. Ali, šta je, zapravo, stvarna prednost vizuelnog nasleđivanja formulara?

Dakle, ovo većinom zavisi od vrste aplikacije koju izrađujete. Ukoliko sadrži veliki broj formulara, od kojih su neki međusobno veoma slični ili jednostavno sadrže uobičajene elemente, tada možete upotrebiti uobičajene komponente i uobičajeno rukovanje događajima u osnovnom formularu i dodati specifično ponašanje i komponente u potklase. Na primer, ukoliko pripremate standardni roditeljski formular sa paletom alata, logotipom, uobičajenim kodom za promenu veličine i zatvaranje, i rukovanjem nekim porukama Windowsa, možete ga upotrebiti kao roditeljski formular za svaki od formulara aplikacije.

Takođe, možete upotrebiti vizuelno nasleđivanje formulara da biste prilagodili aplikaciju za različite klijente, a da ne morate da duplirate kod ili kod definicije formulara; samo nasledite specifične verzije za klijenta iz standardnih formulara. Ne zaboravite da je glavna prednost vizuelnog nasleđivanja to što kasnije možete promeniti originalni formular i automatski ažurirati izvedene formulare. Ovo je poznata prednost nasleđivanja u objektno orijentisanim programskim jezicima. Međutim postoji i koristan sporedan efekat: polimorfizam. Možete dodati virtuelni metod u osnovni formular i zaobići ga u izvedenim formularima. Tada se možete referisati na oba ova formulara i pozvati ovaj metod za svaki od njih.

NAPOMENA

Delphi 5 sadrži nove mogućnosti koje se nazivaju okvirima (Frames), koji podsećaju na vizuelno nasleđivanje formulara. U oba slučaja možete raditi na dve različite verzije formulara prilikom dizajniranja. Ipak, u vizuelnom nasleđivanju formulara Vi definišete dve različite klase (roditeljsku i izvedenu), dok u okviru radite sa klasom i njenom instancom. Okvire ćemo detaljno razmatrati u Poglavlju 4. ■

Nasleđivanje od osnovnog formulara

Pravila koja upravljaju vizuelnim nasleđivanjem formulara su prilično jednostavna, kada jednom jasno shvatite šta je nasleđivanje. U osnovi, izvedeni formular sadrži iste komponente kao i roditeljski formular, i neke nove komponente. Ne možete ukloniti komponente osnovne klase, mada (ukoliko je to vizuelna kontrola) možete da ih učinite nevidljivim. Ono što je važno je da lako možete promeniti svojstva komponenata koje nasleđujete.

Primetićete da, ukoliko promenite svojstvo komponente u izvedenom formularu, bilo kakve izmene istog svojstva u roditeljskom formularu neće imati nikakav efekat. Promena ostalih svojstava komponente će uticati na nasleđene verzije. Možete ponovo sihronizovati vrednosti dva svojstva koristeći komandu lokalnog menija Revert to Inherited iz Object Inspectora. Isti efekat se postiže dodeljivanjem iste vrednosti svojstvima i ponovnim kompajliranjem. Posle izmena više svojstava, možete ih ponovo sinhronizovati na osnovnu verziju pozivanjem komande Revert to Inherited iz lokalnog menija komponente.

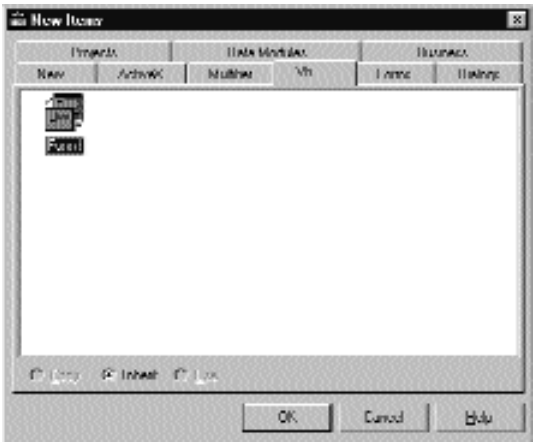
Alternativna tehnika je da otvorite tekstualni opis izvedenog formulara i uklonite linije koje menjaju vrednost svojstva. (Kasnije ćemo pogledati strukturu ovog fajla.) Pored nasleđenih komponenata, novi formular nasleđuje sve metode osnovnog formulara, uključujući rukovanje događajima. Novo rukovanje događajima se može dodati u izvedeni formular, a takođe, možete zaobići postojeće rukovanje događajima.

Da bih opisao kako funkcioniše vizuelno nasleđivanje formulara, ja sam napisao veoma jednostavan primer i nazvao ga VFI. Objasniću Vam korak po korak kako da ga napišete. Prvo, započnite novi projekat i dodajte glavnom formularu četiri kontrole. Zatim odaberite File➥New i odaberite stranu sa nazivom projekta u okviru za dijalog New Items (videti sliku 2.6). Odavde možete odabrati formular iz koga želite da izvedete novi formular. Novi formular će sadržati iste četiri kontrole. Evo inicijalnog tekstualnog opisa novog formulara:

```
inherited Form2: TForm2
  Caption = 'Form2'
end
```

Evo i njegove inicijalne deklaracije klase, odakle se može videti da osnovna klasa nije uobičajena klasa `TForm` već aktuelni osnovni formular:

```
type
  TForm2 = class (TForm1)
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```



SLIKA 2.6 *Okvir za dijalog New Items Vam omogućava da kreirate izvedeni formular*

Primetite u tekstualnom opisu prisustvo ključne reči inherited; takođe, primetite da formular zaista sadrži neke komponente, mada su one definisane u osnovnoj klasi formulara. Ukoliko premestite formular i dodate natpis za jednu od kontrola, tekstualni opis će odslikati promene:

```
inherited Form2: TForm2
  Left = 313
  Top = 202
  Caption = 'Form2'
  inherited Button2: TButton
    Caption = 'Beep . . .'
  end
end
```

Prikazana su samo svojstva sa različitom vrednošću (i uklanjanjem tih svojstava iz tekstualnog opisa izvedenog formulara možete ih vratiti na vrednosti osnovnog formulara, kao što sam to ranije spomenuo). Ja sam, zapravo, promenio natpise većine kontrola kao što se može videti na slici 2.7.

SLIKA 2.7 *Dva formulara primera VFI u vreme izvršavanja*

Svaka od kontrola prvog formulara sadrži rukovanje događajem `OnClick` jednostavnim kodom. Prva kontrola prikazuje izvedeni formular pozivajući metod `Show`; druga i treća kontrola pozivaju proceduru `Beep`; poslednja kontrola prikazuje jednostavnu poruku pozivajući `ShowMessage ('Hi')`.

Šta se dešava u izvedenom formularu? Prvo, potrebno je da uklonimo kontrolu Show jer je drugi formular već vidljiv. Ipak, ne možemo kontrolu potpuno ukloniti u izvedenom formularu. Alternativno rešenje je da ostavimo komponentu, ali da za svojstvo `Visible` odaberemo `False`. Kontrola će još uvek postojati, ali neće biti vidljiva (kao što možete pretpostaviti kada pogledate sliku 2.7). Preostale tri kontrole će biti vidljive, ali će imati različito rukovanje događajima. To je jednostavno postići. Ukoliko odaberete događaj `OnClick` kontrole u izvedenom formularu (dva puta kliknite događaj), prikazaće se prazan metod koji se malo razlikuje od unapred određenog metoda:

```
procedure TForm2.Button2Click (Sender: TObject);
begin
  inherited;
end;
```

Ključna reč `inherited` označava poziv odgovarajućem rukovanju događajem osnovnog formulara. Ovu ključnu reč uvek dodaje Delphi, iako rukovanje nije definisano u roditeljskoj klasi (što meni ne izgleda kao dobra ideja). Veoma je jednostavno izvršiti kod osnovnog formulara i obaviti dodatne operacije:

```
procedure TForm2.Button2Click (Sender: TObject);
begin
  inherited;
  ShowMessage ('Hi');
end;
```

Ovo nije jedina mogućnost. Alternativni pristup je da napišete potpuno novo rukovanje događajem i da ne izvršite kod osnovne klase, kao što sam ja to učinio za treću kontrolu primera VFI:

```
procedure TForm2.Button3Click (Sender: TObject);
begin
       ShowMessage ('Hi');
end;
```

Još jedna mogućnost se zasniva na pozivu metoda osnovne klase posle izvršenja novog koda, pozivajući ga kada se ispuni uslov, ili pozivajući neko drugo rukovanje događajem osnovne klase, kao što sam ja to učinio za četvrtu kontrolu:

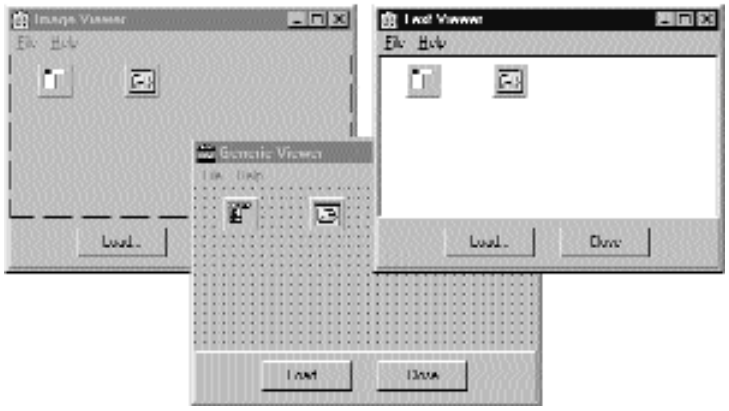
```
procedure TForm2.Button4Click (Sender: TObject);
begin
  inherited Button3Click (Sender);
  inherited;
end;
```

Vi verovatno ovo nećete često činiti, ali je potrebno da znate da to možete učiniti. Naravno, možete svaki metod osnovnog formulara smatrati metodom novog formulara i slobodno ga pozvati. Ovaj primer Vam omogućava da istražite neke mogućnosti vizuelnog nasleđivanja formulara, ali da biste videli stvarnu moć, potrebno je da pogledate primere iz svakodnevne prakse koji su mnogo složeniji nego što ova knjiga ima prostora da ih prikaže. Postoji još nešto što želim ovde da Vam pokažem: vizuelni polimorfizam formulara (visual form polymorphism).

Polimorfni formulari

Problem je jednostavan. Ukoliko dodate rukovanje događajem formularu, a zatim ga izmenite u izvedenom formularu, ne postoji način da se obratite dvama metodima koristeći uobičajenu promenljivu osnovne klase jer rukovanja događajima koriste statičko povezivanje po definiciji.

Zbunjujuće? Evo primera koji je namenjen iskusnim programerima Delphija. Pretpostavimo da želite da izradite formular za pregled bitmapa i formular za pregled teksta u istom programu. Dva formulara sadrže slične elemente, slične palete alata, komponentu OpenDialog i različite komponente za pregled podataka. Dakle, odlučili ste da izradite osnovni formular koji sadrži zajedničke elemente i iz njega izvedete dva formulara. Ta tri formulara u vreme dizajniranja možete videti na slici 2.8.



SLIKA 2.8 *Osnovni formular i dva različita izvedena formulara u vreme dizajniranja primera PoliForm*

Evo tekstualnog opisa glavnog formulara:

```
object ViewerForm: TViewerForm
  Caption = 'Generic Viewer'
  Menu = MainMenu1
  object Panel1: TPanel
    Align = alBottom
    object ButtonLoad: TButton. . .
    object CloseButton: TButton...
  end
  object MainMenu1: TMainMenu
    object File1: TMenuItem...
      object Load1: TMenultem...
      object N1: TMenultem...
      object Close1: TMenultem...
    object Help1: TMenuitem...
      object AboutPoliForm1: TMenultem...
  end
  object OpenDialog1: TOpenDialog...
end
```

Dva izvedena formulara se samo malo razlikuju, ali predstavljaju novu komponentu, bilo pregled slika (`TImage`) bilo pregled teksta (`TMemo`):

```
inherited ImageViewerForm: TImageViewerFarm
  Caption = 'Image Viewer'
  object Image1: TImage [0]
    Align = alClient
  end
  inherited OpenDialog1: TOpenDialog
    Filter = 'Bitmap file/*.bmp/Any file/*.*'
  end
end
inherited TextViewerForm: TTextViewerForm
  Caption = 'Text Viewer'
  object Memo1: TMemo [1]
    Align = alClient
```

```
end
inherited OpenDialog1: TOpenDialog
    Filter = 'Text files/*.txt/Any file /*.*'
  end
end
```

Glavni formular sadrži neki zajednički kod. Kontrola Close i komanda File➥Close pozivaju metod Close formulara. Komanda Help➥About prikazuje jednostavan okvir za dijalog sa porukom. Kontrola Load osnovnog formulara sadrži sledeći kod:

```
procedure TViewerForm.ButtonLoadClick(Sender: TObject);
begin
  ShowMessage ('Error: File loading code missing');
end;
```

Komanda File➥Load umesto toga poziva drugi metod:

```
procedure TViewerForm.Load1Click(Sender: TObject);
begin
  LoadFile;
end;
```

Ovaj metod je definisan u klasi TViewrForm kao

```
public
  procedure LoadFile; virtual; abstract;
```

S obzirom na to da je ovo apstraktni metod, mi ćemo ga ponovo definisati (i zaobići) u izvedenim formularima:

```
type
  TImageViewerForm = class(TViewerForm)
    Image1: TImage;
    procedure ButtonLoadCl ick(Sender: TObject);
  public
    procedure LoadFile; override;
  end;
```

Kod ovog jednostavnog metoda LoadFile koristi komponentu OpenDialog da bi od korisnika zatražio da odabere ulazni fajl i da ga učita u komponentu za slike:

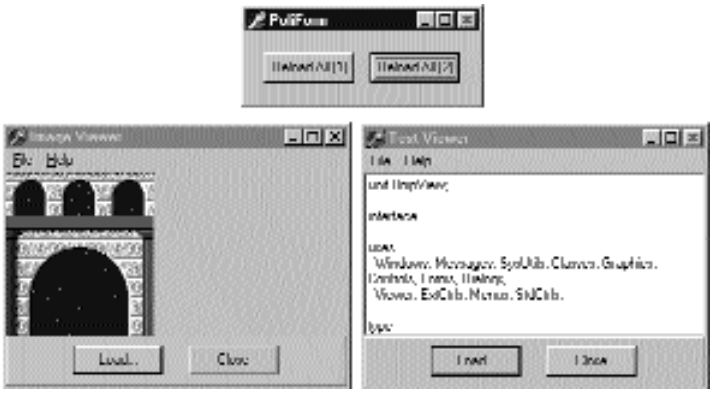
```
procedure TImageViewerForm.LoadFile;
begin
  if OpenDialogl.Execute then
    Image1.Picture.LoadFromFile (
      OpenDialog1.Filename);
end;
```

Druga izvedena klasa sadrži sličan kod, učitavajući tekst u komponentu za tekst. Projekat sadrži još jedan formular, glavni formular sa dve kontrole, koji se koristi za ponovno učitavanje fajlova u svaki od formulara za prikaz. Glavni formular je jedini formular koji se kreira kada projekat započne izvršavanje. Generički formular za prikaz se nikada ne kreira: on predstavlja samo generičku osnovnu klasu koja sadrži zajednički kod i komponente dve potklase. Formulari dve potklase se kreiraju događajem OnCreate glavnog formulara:

```
procedure TMainForm.FormCreats(Sender: TObject);
var
  I: Integer;
begin
  FormList [1] := TTextViewerForm.Create (Application);
  FormList [2] := TImageViewerForm.Create (Application);
  for I := 1 to 2 do
    FormList[I].Show;
end;
```

Pogledajte sliku 2.9 da biste videli rezultujuće formulare (kada su slika i tekst već učitani). `FormList` je polimorfni niz formulara deklarisan u klasi `TMainForm` kao

```
private
  FormList: array [1..2] of TViewerForm;
```



SLIKA 2.9 *Primer PoliMorph u vreme izvršavanja*

Primetite da je potrebno da dodate jedinicu View (ali ne i određene formulare) u klauzulu `uses` interfejs dela glavnog formulara da biste načinili ovu deklaraciju u klasi. Niz formulara se koristi za učitavanje novog fajla u svakom od formulara za prikaz kada se klikne na jednu od kontrola. Rukovanje događajem `OnClick` ove dve kontrole koristi različite pristupe:

```
procedure TMainForm.ReloadButton1Click (Sender: TObject);
var
  I: Integer;
begin
  for I := 1 to 2  do
    FormList [I] .ButtonLoadClick (self);
end;

procedure TMainForm.ReloadButton2Click(Sender: TObject);
var
  I: Integer;
begin
  for I := 1 to 2 do
    FormList [I] .LoadFile;
end;
```

Druga kontrola jednostavno poziva virtuelni metod i funkcionisaće bez problema. Prva kontrola poziva rukovanje događajem i uvek će pozvati generičku klasu `TFormView` (prikazujući poruku o grešci metoda `ButtonLoadClick`). Ovo se dešava jer je ovaj metod statički a ne virtuelni.

Postoji li način da ovaj pristup funkcioniše? Naravno. Deklarišite metod `ButtonLoadClock` klase `TFormView` kao virtuelni i deklarišite ga kao zaobiđeni (overridden) u svakoj od izvedenih klasa kao što smo to učinili za bilo koji drugi virtuelni metod:

```
type
  TViewerForm = class(TForm)
    // components and plain methods...
    procedure ButtonLoadClick(Sender: TObject); virtual;
  public
    procedure LoadFile; virtual; abstract;
  end;
. . .
type
  TImageViewerForm = class(TViewerForm)
    Image1: TImage;
    procedure ButtonLoadClick(Sender: TObject); override;
  public
    procedure LoadFile; override;
  end;
```

Jednostavno, zar ne? Trik zaista funkcioniše mada nikada nije pomenut u Delphijevoj dokumentaciji. Mogućnost upotrebe virtuenog rukovanja događajima je ono što ja, zapravo, podrazumevam pod vizuelnim polimorfizmom formulara.

Šta je sledeće?

U ovom poglavlju smo razmatrali osnove objektno orijentisanog programiranja u Object Pascalu. Razmotrili smo definiciju klasa, upotrebu metoda, enkapsulaciju, nasleđivanje, polimorfizam i tip informacija u vreme izvršavanja. To je stvarno dosta informacija ukoliko ste početnik, ali ukoliko ste upoznati sa drugim OOP jezicima, ili ste već koristitili prethodne verzije Delphija, trebalo bi da, u Vašem programiranju, možete da upotrebite elemente koje smo prikazali u ovom poglavlju.

Naredno poglavlje nastavlja razmatranje načina kako Delphi implementira OOP. Ono objašanjava ostale mogućnosti jezika, kao što su pokazivači metoda, reference klase, svojstva, događaji i izuzeci, koje su naročito važne za Delphi podršku vizuelnom stilu programiranja. Poglavlje 3 Vam, takođe, pokazuje kako da definišete sopstvene komponente. Poglavlje 4 se zatim bavi strukturom VCL-a (Visual Component Library) i razmatra nekoliko važnih klasa.

Razumevanje tajni Object Pascala i strukture VCL-a je od vitalnog značaja da biste postali odličan programer u Delphiju. Ove teme čine osnove rada sa VCL-om; kada ih upoznamo u naredna dva poglavlja, preći ćemo, konačno, na Deo II ove knjige da bismo se pozabavili programiranjem svakodnevnih aplikacija koristeći sve različite komponente koje sadrže Delphi.

POGLAVLJE

3

Unapređeni Object Pascal

U PRETHODNOM POGLAVLJU STE VIDELI OSNOVE JEZIKA OBJECT PASCAL KOJI KORISTI DELPHI: KLASE, OBJEKTE, METODE, KONSTRUKTORE, NASLEĐIVANJE, KASNO POVEZIVANJE I TIP INFORMACIJA U VREME IZVRŠAVANJA. SADA JE POTREBNO DA KRENEMO KORAK DALJE I UPOZNAMO NEKE NAPREDNIJE MOGUĆNOSTI JEZIKA. NEKA OD PROŠIRENJA KOJA ĆEMO RAZMATRATI U OVOM POGLAVLJU, NAROČITO KLJUČNA REČ PUBLISHED, SVOJSTVA I DOGAĐAJI, SU STRIKTNO VEZANA ZA DELPHIJEV VIZUELNI PROGRAMSKI MODEL. ZAPRAVO, DOK BUDEMO RAZMATRALI OVE ELEMENTE, JA ĆU VAM POKAZATI KAKO DA KREIRATE JEDNOSTAVNU KOMPONENTU.

Neki drugi elementi Object Pascala, kao što su izuzeci i interfejsi, nisu tako blisko povezani sa vizuelnim elementima Delphija. Ipak je važno da ih razumete, kao i nekoliko drugih elemenata koje razmatramo u ovom poglavlju, da biste mogli da napišete korektan kod Delphi aplikacije.

Metodi klase i podaci klase

Kada polje definišete u klasi, Vi zapravo određujete da polje treba dodati svakoj instanci objekta te klase. Svaka instanca ima svoju nezavisnu reprezentaciju (na koju se možete pozvati ključnom reči `Self`). U nekim slučajevima, ipak, može da bude korisno postojanje polja koje dele svi objekti klase.

Drugi objektno orijentisani jezici sadrže formalne konstrukte kojima se ovo izražava, kao što je ključna reč `static` u jeziku C++. Međutim, u Object Pascalu ovu mogućnost možemo simulirati enkapsulacijom na nivou jedinice. Možete jednostavno dodati promenljivu u deo jedinice `implementation` da biste obezbedili promenljivu klase — jednu memorijsku lokaciju koju će deliti svi obejkti klase.

Ukoliko je potrebno da ovoj vrednosti pristupite van jedinice, možete upotrebiti metod klase. Ipak, ovo Vas primorava da upotrebite ovaj metod na jednu od instanci klase. Alternativno rešenje je da deklarišete metod klase (class method). Metod klase ne može pristupiti podacima bilo kog pojedinačnog objekta, ali se može primeniti na celu klasu umesto na određenu instancu klase. Metod klase je povezan sa klasom, ne sa njenim instancama ili objektima (nešto kao `static` funkcija članica u jezicima Java i C++).

Da biste deklarisali metod klase u Object Pascalu, jednostavno dodajte ključnu reč `class` ispred metoda:

```
type
  MyClass = class
    class function ClassMeanValue: Integer;
```

Upotreba metoda klase nije česta u Object Pascalu jer isti efekat možete postići dodavanjem procedure ili funkcije jedinici koja deklariše klasu. Međutim, objektno orijentisani čistunci će definitivno više voleti da upotrebe metod klase umesto rutine koja nema veze sa klasom. Zapravo, VCL veoma često koristi metode klase, mada postoji i veliki broj globalnih podrutina. Primetićete da u Delphiju metodi klase mogu da budu virtuelni, te se mogu zaobići i koristiti da bi se postigao polimorfizam.

Klasa sa brojačem objekata

Kada se podaci jedinice koriste za održavanje opštih informacija koje imaju veze sa klasom (kao što je broj objekata koji su kreirani ili spisak tih objekata), možete upotrebiti metode klase da biste pristupili tim podacima. To je ono što naredni primer upravo radi.

Program CountObj je proširenje primera CreateC iz prethodnog poglavlja. Formular je još uvek prilično prazan, ali sam ja dodao novi kod. Preciznije, ja sam dodao potpuno novu klasu, koja je izvedena iz VCL klase `TButton` i koja dodaje novu mogućnost, očigledno za prebrojavanje objekata. Evo deklaracije nove klase:

```
type
  TCountButton = class (TButton)
    constructor Create (AOwner: TComponent); override;
    destructor Destroy; override;
    class function GetTotal: Integer;
end;
```

NAPOMENA

Ono što ovde možete videti je potpuno funkcionalna komponenta. U ovom slučaju nećemo je registrovati i nećemo je dodati Delphijevoj paleti komponenata, iako to nije naročito teška operacija. Prilagođavanje postojećih komponenata može da bude prilično jednostavno. Ovu temu ćemo razmatrati nešto kasnije u ovom poglavlju, a detaljno ćemo je razmatrati u Poglavlju 13. ■

Svaki put kada se kreira objekat, program uvećava brojač pre poziva konstruktora osnovne klase. Svaki put kada se objekat ukloni, brojač se umanjuje:

```
constructor TCountButton.Create (AOwner: TComponent);
begin
  inherited Create (AOwner);
  Inc (TotBtns);
end;

destructor TCountButtonDestroy;
begin
  Dec (TotBtns);
  inherited Destroy;
end;
```

Brojač je promenljiva koja je deklarisana u delu jedinice `implementation`, te joj se kao takvoj ne može pristupiti van jedinice. Samo nam metod klase omogućava da pročitamo trenutnu vrednost promenljive. Možete direktno inicijalizovati ovu promenljivu kada je definisana:

```
implementation

var
  TotBtns: Integer = 0;

class function TCountButton.GetTotal: Integer;
begin
  Result := TotBtns;
end;
```

Sada možemo kreirati objekte ovog novog tipa malom promenom koda metoda `FormMouseDown`:

```
begin    with TCountButton.Create (Self) do
  begin
    Parent := Self;
    // same code as before...
```

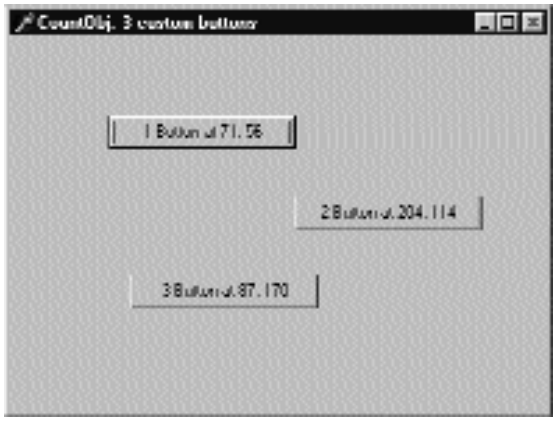
Svaki put kada se kreira objekat `TCountButton`, trenutni broj objekata se prikazuje na početku natpisa. Možemo pozvati metod klase `GetTotal` za novokreirani objekat (primetićete da smo unutar iskaza `with`), baš kao što bismo pozvali bilo koji drugi metod. Ipak, možemo pozvati isti metod, a da nemamo validnu instancu objekta. To je ono što činimo kada istekne vreme tajmera koji sam dodao formularu:

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Caption := Format ('CountObj: %d custom buttons',
    [TCountButton.GetTotal]);
end;
```

Svojstvo `Caption` se u ovom kodu odnosi na natpis formulara. Na slici 3.1 možete videti efekat ovog poziva. Nedostatak ovog primera je taj da možemo samo kreirati objekte, ali ih ne možemo ukloniti, tako da vidimo da se ukupan broj postojećih objekata uvek uvećava i da se vrednost nikada ne umanjuje.

Da bismo videli da se broj objekata koji postoje kreće ka nuli, možemo pokušati da proverimo broj objekata posle uklanjanja formulara zajedno sa objektima `TCountButton` koje poseduje.

```
finalization
  MessageBox (0, PChar (Format (
    'There are %d CountButton objects',
    [TCountButton.GetTotal])),
    'Finalization', mb_OK);
end.
```



SLIKA 3.1 *Izlaz primera CountObj posle kreiranja nekoliko objekata TCountButton*

UPOZORENJE

U kodu `finalization` iznad ja sam morao da upotrebim Windows API funkciju (`MessageBox`) umesto procedure Delphija (kao što je `ShowMessage`). Razlog je što se kod `finalization` jedinice izvršava pošto se uklone neki Delphi globalni objekti te je bolje da se ne oslonim na njih. ■

Program jednostavno prikazuje Windowsov okvir za poruke dajući broj objekata koji postoje, vrednost koja se dobija pozivom metoda klase `GetTotal`. Ukoliko pokrenete program, broj na izlazu je nula, mada moram da kažem da to nije sigurno, ali je tako zbog redosleda po kojem se objekti uklanjaju. Kompajler koristi specifični poredak za inicijalizaciju i finalizaciju jedinica: počevši od izvornog koda projekta, jedinice na koje se referiše se inicijalizuju pre, a finalizuju pošto se na njih referiše. Tipično, projekat će inicijalizovati jedinicu Forms, koja će odmah inicijalizovati ostale VCL

jedinice, a zatim će inicijalizovati Vašu jedinicu formulara, koja će prvo inicijalizovati jedinice koje opisuju komponente koje koristite (to jest, one koje se nalaze u iskazu `uses`).

Ipak, na prvi pogled, nije lako odrediti u ovom redosledu događaja kada će glavni formular aplikacije i njegove komponente biti uklonjeni. Da bismo proverili da li sve zaista funkcioniše, ja sam dodao kod koji poziva `MessageBox` u rukovanju događajem formulara `OnDestroy`, koji se poziva pre nego što se ukloni formular.

Ukoliko pokrenete program, videćete da kada se izvrši metod `FormDestroy`, svi objekti koje ste kreirali još uvek postoje: ali, odmah posle toga, objekti se uklanjaju, a vrednost bojača se vraća na nulu. Videćemo složeniji primer u kojem ćemo ukloniti kontrole u vreme izvršavanja, posle razmatranja pokazivača metoda u dodatku "Unapređeni primer brojača".

Pokazivači metoda

Još jedan dodatak Delphija jeziku Object Pascal je koncept pokazivača metoda (method pointers). Tip pokazivača metoda je sličan proceduralnom tipu, ali se referiše na metod. Tehnički, tip pokazivača metoda je proceduralni tip koji sadrži implicitni parametar `Self`. Drugim rečima, pokazivač metoda čuva dve adrese: adresu koda metoda i adresu instance objekta (podataka). Adresa instance objekta će se prikazati kao `Self` unutar tela metoda kada se kod metoda pozove upotrebom pokazivača metoda. To objašnjava definiciju Delphijevog generičkog tipa `TMethod`, sloga koji sadrži polje `Code` i polje `Data`.

Deklaracija tipa pokazivača metoda je slična deklaraciji proceduralnog tipa, izuzev što sadrži ključne reči `of object` na kraju deklaracije:

```
type
  IntProceduralType = procedure (Num: Integer);
  IntMethodPointerType = procedure (Num: Integer) of object;
```

Kada ste deklarisali pokazivač metoda, kao što je ovaj iznad, možete deklarisati promenljivu ovog tipa i dodeliti joj kompatibilan metod drugog objekta. Šta je to kompatibilan metod? To je metod koji sadrži iste parametre kao što su parametri tipa pokazivača metoda, recimo jedan parametar Integer i prethodnom primeru.

Na prvi pogled, cilj ove tehnike nije odmah jasan, ali on predstavlja temelj Delphijeve tehnologije komponenata. Tajna leži u reči delegacija (delegation). Ukoliko je neko izradio objekat koji sadrži pokazivače metoda, Vi slobodno možete promeniti ponašanje objekta dodeljivanjem novih metoda pokazivačima. Zvuči li Vam ovo poznato? Trebalo bi da Vam zvuči.

Kada dodate rukovanje događajem kontrole `OnClick`, Delphi čini upravo to. Kontrola ima pokazivač metoda, nazvan `OnClick`, a Vi mu možete dodeliti metod formulara direktno ili indirektno. Kada korisnik klikne kontrolu, ovaj metod se izvršava, iako ste ga definisali unutar druge klase (tipično u formularu).

Ovo što sledi je listing koji skicira kod koji Delphi zapravo koristi, da bi definisao rukovanje događajem komponente kontrole i odgovarajućeg metoda formulara:

```
type
  TNotifyEvent = procedure (Sender: TObject) of object;

  MyButton = class
    OnClick: TNotifyEvent;
  end;

  TForm1 = class (TForm)
    procedure ButtonClick (Sender: TObject);
    Button1: MyButton;
  end;

var
  Form1: TForm;
```

Sada unutar procedure možete napisati:

```
MyButton.OnClick := Form1.ButtonClick;
```

Jedina stvarna razlika između ovog fragmenta koda i VCL koda je da je `OnClick` naziv svojstva, a da se stvarni podaci na koje se poziva nazivaju `FOnClick`. Događaj koji se prikazuje na strani Events Object Inspectora, zapravo, nije ništa drugo do svojstvo tipa pokazivača metoda.

To, na primer, znači da možete dinamički izmeniti rukovanje događajem koje je pridodato komponenti u vreme dizajniranja, ili čak načiniti novu komponentu u vreme izvršavanja i dodeliti joj rukovanje događajem. Na primer, možemo dodati formularu primera Counter sledeći metod:

```
procedure TForm1.ButtonClick (Sender: TObject);
begin
  ShowMessage ('Button preseed');
end;
```

a zatim za svaku novokreiranu kontrolu napisati naredni kod:

```
with TCountButton.Create (Self) do
begin
  OnClick := ButtonClick;
```

Ovim kodom, svaka od kontrola će reagovati kada kliknete mišem, tako što će prikazati poruku, jer sve komponente dele isto rukovanje događajem. Ipak, možemo upotrebiti parametar događaja `Sender` da bismo prilagodili poziv za svaku od kontrola. To je upravo ono što ću učiniti u primeru koji razmatram u dodatku "Unapređeni primer Counter", koji predstavlja još kompletnije proširenje programa Counter.

UNAPREĐENI PRIMER COUNTER

Sada kada znamo kako da upotrebimo pokazivače metoda, možemo unaprediti primer CountObj koristeći pokazivače metoda. Naziv novog primera je CountObj2. Njegova svrha je dodavanje rukovanja događajem `OnKeyPress` novih objekata koje korisnik dinamički kreira. Dodajte naredni kod u deklaraciju klase formulara:

```
procedure ButtonKeyPress (Sender: TObject; var Key: Char);
```

Parametri su onakvi kakve zahteva događaj ovog tipa. Ukoliko odaberete događaj `OnKeyPress` komponente formulara i pritisnete taster F1 da biste pozvali Help, pronaći ćete sledeću deklaraciju:

```
TKeyPressEvent = procedure (Sender: TObject;
  var Key: Char) of object;
property OnKeyPress: TKeyPressEvent;
```

Kao što možete videti u poslednjoj liniji, događaj se zasniva na tipu pokazivača metoda `TKeyPressEvent` koji je naveden u liniji iznad. Zbog toga je potrebno da napišemo metod koji se kompajlira sa ovim tipom pokazivača metoda, kao onaj koji je predstavljen u prethodnom odeljku.

Da bismo povezali ovaj metod sa događajem `OnKeyPress` kontrola koje dinamički kreiramo, potrebna nam je samo jedna linija koda u metodu `FormMouseDown`:

```
with TCountButton.Create (Self) do
begin
  . . .
  // set the event handler
  OnKeyPress := ButtonKeyPress;
  // grab the input focus
  SetFocus;
end;
```

Druga linija koda prebacuje fokus na novokreiranu kontrolu tako da je sledeći ulaz sa tastature upućen na kontrolu.

Sada možemo napisati kod metoda `ButtonKeyPress`. Pritisnite kombinaciju tastera Ctrl+Shift+C da biste aktivirali Delphi Code Completion, a zatim popunite deklaraciju metoda stvarnim kodom. Pošto se ulaz sa tastature šalje kontroli koja ima ulazni fokus, možete jednostavno kliknuti kontrolu ili upotrebiti taster Tab da biste odabrali kontrolu koju želite da uklonite; zatim pritisnite taster Backspace.

Prvi pristup koji sam isprobao prilikom razvoja ove aplikacije, bilo je jednostavno uklanjanje objekta koji je prosleđen kao parametar `Sender`, a to je objekat koji je primio događaj.

```
procedure TForml.ButtonKeyPress(Sender: TObject;
  var Key: Char);
begin
  if Key = #8 then
    Sender.Free
end;
```

Ovaj kod poziva izuzetak. Ne možemo ukloniti objekat dok obrađujemo neki od njegovih događaja. Umesto toga moramo odložiti njegovo uklanjanje. U osnovi postoje dva pristupa. Možemo sačuvati objekat koji želimo da uklonimo u privatnom polju klase formulara, i zatim ga kasnije ukloniti unutar koda koji se periodično aktivira tajmerom. Ovaj kod možete videti u primeru CountOld. Primetićete da upotreba tajmera prouzrokuje malu grešku u programu: ukoliko se tasteri Backspace obrađuju pre nego što se pokrene tajmer, samo jedna kontrola će biti uklonjena.

Drugim pristupom, implementiranim u primeru CountObj2, šalje se Windows poruka (recimo `wm_User`) formularu upotrebom API funkcije `PostMessage`. Time se prouzrokuje odlaganje, jer poruka treba da stigne do prozora, a biće prihvaćena i obrađena posle izvršavanja rukovanja događajem. Da bismo upotrebili drugi pristup, možemo napisati sledeće rukovanje događajem `OnKeyPress` za svaku od novih kontrola:

```
procedure TFormI.ButtonKeyPress(Sender: TObject;
  var Key: Char);
begin
  // if user pressed backspace
  if Key = #8 then
  begin
    // set this as the object to destroy
    ToDestroy := Sender as TButton;
    // post message to perform destruction
    PostMessage (Handle, wm_User, 0, 0);
  end;
end;
```

U ovom kodu `ToDestroy` je privatno polje formulara tipa podataka `TButton`. Polje automatski dobija vrednost `nil` (nema objekta koji treba ukloniti) kada se formular kreira (to je unapred određena inicijalizacija polja klase). Kada korisnik pritisne taster Backspace, aktuelna kontrola (`Sender` metoda `ButtonKeyPress`) se čuva u polju `ToDestroy`. U ovom trenutku Windows API poziv `PostMessage` šalje poruku aktuelnom prozoru (identifikovanom vrednošću svojstva `Handle`). Rukovanje ovom porukom je definisano u klasi formulara na sledeći način:

```
type
  TForml = class(TForm)
    . . .
  private
    ToDestroy: TButton;
  public
    procedure WmUser (var Msg: TMessaqe); message wm_User;
```

Sada možemo pogledati kod ovog metoda, u kome program može proveriti da li postoji kontrola koju treba ukloniti pre uklanjanja i dodeliti joj `nil`:

```
procedure TForml.WmUser (var Msg: TMessage);
begin
  // if there is an object to destroy
  if Assigned (ToDestroy) then
  begin
    // moves the input focus to the next control
    SelectNext (ToDestroy, True, True);
    // destroy the object and set the reference to nil
    FreeAndNil (ToDestroy);
  end;
  // update the form caption
  Caption := Format ('CountObj: %d custom buttons',
    [TCountButton GetTotal]);
end;
```

Da bi se program ponašao malo bolje pre uklanjanja objekta, ja sam ulazni fokus pomerio na narednu kontrolu pozivanjem metoda `SelectNext`. Program zatim poziva novu proceduru Delphija 5, `FreeAndNil`, koja poziva metod `Free` objekta, koji odmah poziva destruktor `Destroy`. Kako je destruktor virtuelan, program poziva detruktor klase `TCountButton`, koji umanjuje brojač objekata. Ja sam, zbog ovoga, kod koji uklanja objekat smestio pre koda koji ažurira natpis. Pre poziva `Free`, `FreeAndNil` referenci `ToDestroy` dodeljuje `nil`.

Više o oslobađanju objekata i manipulisanju memorijom ćete naći u odeljku "Objekti i memorija" kasnije u ovom poglavlju.

Reference klase

Pošto smo se upoznali sa nekoliko tema vezanih za metode, sada možemo preći na temu referenci klasa (class references) i još više unaprediti naš primer dinamičkog kreiranja komponenata. Prva stvar koju treba imati na umu je da referenca klase nije klasa, nije objekat i nije referenca na objekat; to je jednostavno referenca tipa klase.

Tip reference klase određuje tip promenljive reference klase. Zvuči zbunjujuće? Nekoliko linija koda bi moglo da razjasni stvari. Pretpostavimo da je definisana klasa `TMyClass`. Sada možete definisati novi tip reference klase, koji se odnosi na tu klasu:

```
type
  TMyClassRef = class of TMyClass;
```

Sada možete deklarisati promenljive ovih tipova. Prva promenljiva se odnosi na objekat, druga se odnosi na klasu:

```
var
  AClassRef: TMyClassRef;
  AnObject: TMyClass;
begin
  AClassRef := TMyClass;
  AnObject := TMyClass.Create;
```

Možda se pitate čemu služe reference klase. Uopšte uzev, reference klase Vam omogućavaju da manipulišete tipom podataka klase u vreme izvršavanja. Možete upotrebiti referencu klase u bilo kom izrazu u kojem je dozvoljena upotreba tipa podataka. Zapravo, ne postoji mnogo takvih izraza, ali nekoliko slučajeva je interesantno. Najjednostavniji primer je kreiranje objekta. Prethodne dve linije možemo da napišemo i ovako:

```
AClassRef := TMyClass;
AnObject := AClassRef.Create;
```

Ovoga puta sam upotrebio konstruktor `Create` nad referencom klase umesto nad aktuelnom klasom; ja sam koristio referencu klase da bih kreirao objekat te klase.

NAPOMENA

Reference klasa nas podsećaju na koncept metaklasa koji postoji u drugim OOP jezicima. U Object Pascalu, međutim, referenca klase nije klasa već samo pokazivač tipa. Zbog toga analogija sa metaklasama (klase koje opisuju druge klase) nije ispravna. Zapravo, `TMetaclass` je takođe termin koji koristi Borland C++ Builder. ■

Tipovi reference klase ne bi bili korisni kada ne bi podržavali pravilo kompatibilnosti tipova koje se odnosi na tipove klase. Kada deklarišete promenljivu reference klase, recimo `MyClassRef`, možete joj dodeliti tu specifičnu klasu i bilo koju od potklasa. Ukoliko je `MyNewClass` potklasa moje klase, možete takođe napisati:

```
AClassRef := MyNewClass;
```

Delphi deklariše dosta referenci klase u izvršnoj biblioteci i VCL-u uključujući sledeće:

```
TClass = class of TObject;
ExceptClass = class of Exception;
TComponentClass = class of TComponent;
TControlClass = class of TControl;
TFormClass = class of TForm;
```

Tip reference klase `TClass` se može koristiti za čuvanje reference na bilo koju klasu koju napišete u Delphiju, jer je svaka klasa, na kraju krajeva, izvedena iz klase `TObject`. Referenca `TFormClass` se koristi u izvornom kodu većine Delphi projekata. Metod `CreateForm` objekta `Application`, zapravo, zahteva kao parametar klasu formulara za kreiranje:

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

Prvi parametar je referenca klase, drugi je promenljiva koja čuva referencu na kreiranu instancu objekta.

Konačno, kada imate referencu klase, možete je primeniti na metode klase. Imajući na umu da je svaka klasa izvedena iz klase `TObject`, na svaku referencu klase možete primeniti neke metode klase `TObject`, uključujući `InstanceSize`, `ClassName`, `ParentClass` i `InheritsFrom`. U narednom poglavlju ću razmatrati ove metode klase i druge metode klase `TObject`.

Kreiranje komponenata upotrebom referenci klase

Kakva je *praktična* upotreba referenci klase u Delphiju? Mogućnost manipulisanja tipom podataka u vreme izvršavanja predstavlja osnovni element Delphi okruženja. Kada formularu dodate novu komponentu, tako što ćete ga izabrati sa Component Palette, Vi odabirate tip podataka i kreirate objekat tog tipa podataka. (Zapravo, to je ono što Delphi radi za Vas iza scene.)

Da biste imali bolju ideju kako reference klase funkcionišu, ja sam napravio jednostavan primer koji sam nazvao ClassRef. Formular koji prikazuje ovaj primer je zaista jednostavan. Sadrži tri opcione kontrole (radio buttons) koje se nalaze na panelu u gornjem delu formulara. Kada odaberete jednu od kontrola i kliknete formular, moći ćete da kreirate nove komponente jednog od tri tipa koje su naznačene oznakama kontrola: opcione kontrole, kontrole (push button) i polja za izmene (edit boxes).

Da bi se program pravilno izvršavao, potrebno je da izmenite nazive trima komponentama. Formular, takođe, mora sadržati polje reference klase:

```
private
  ClassRef: TControlClass;
  Counter: Integer;
```

Prvo polje čuva novi tip podataka svaki put kada korisnik klikne jednu od opcionih kontrola. Evo jednog od tri metoda:

```
procedure TForm1.RadioButtonRadioClick (Sender: TObject);
begin
  ClassRef := TRadioButton;
end;
```

Preostale dve opcione kontrole sadrže rukovanje događajem `OnClick` slično prethodnom, dodeljujući vrednost `TEdit` ili `TButton` polju `ClassRef`. Slično dodeljivanje postoji u rukovanju događaja `OnCreate` formulara koje se koristi kao metod za inicijalizaciju.

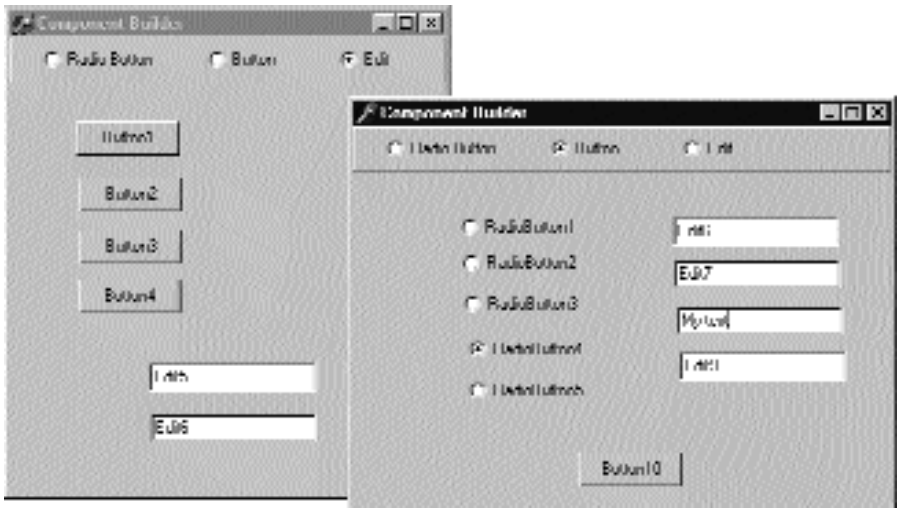
Interesantan deo koda se izvršava kada korisnik klikne formular. Ponovo, ja sam odabrao metod formulara `OnMouseDown` da bih dobio poziciju na kojoj je kliknuto mišem:

```
procedure TForml. FormMouseDown(
  Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  NewCtrl : TControl;
  MyName: String;
begin
  // create the control
  NewCtrl := ClassRefCreate (Self);
  // hide it temporarily, to avoid flickering
  NewCtrl.Visible := False;
  // set parent and position
  NewCtrl.Parent := Self;
  NewCtrl.Left := X;
  NewCtrl.Top := Y;
  // compute the unique name (and caption)
  Inc (Counter);
  MyName := ClassRef.ClassName + IntToStr (Counter);
  Delete (MyName, 1, 1);
  NewCtrl.Name := MyName;
  // now show it
  NewCtrl.Visible := True;
end;
```

Prva linija koda ovog metoda je ključna. Njome se kreira novi objekat tipa podataka klase koji se čuva u polju `ClassRef`. Ovo smo postigli jednostavnim dodeljivanjem konstruktora `Create` referenci klase. Sada možete dodeliti vrednost svojstvu `Parent`, odrediti poziciju nove komponente, dodeliti joj naziv (koji se automatski koristi kao svojstvo `Caption` ili `Text`) i učiniti je vidljivom.

Obratite pažnju na kod koji se koristi za imenovanje; da bi se imitirala unapred određena Delphi konvencija imenovanja, ja sam upotrebio naziv klase uz izraz `ClassRef.ClassName`, koristeći metod klase od klase `TObject`. Zatim sam dodao broj na kraj naziva i uklonio sam početno slovo stringa. Za prvu opcionu kontrolu osnovni string je `TRadioButton` i 1 na kraju, a kada oduzmem T na početku naziva klase — *RadioButton1*. Zvuči poznato?

Na slici 3.2 možete videti dva izlaza ovog programa. Primetićete da imenovanje nije identično imenovanju koje koristi Delphi. Delphi koristi zaseban brojač za svaki tip kontrole; ja sam koristio samo jedan brojač za sve komponente. Ukoliko postavite opcionu kontrolu, kontrolu i polje za izmene na formular primera ClassRef, njihovi nazivi će biti *RadioButton1*, *Button2* i *Edit3*.



SLIKA 3.2 *Dva primera izlaza primera ClassRef, u dva zasebna prozora*

Objekti i memorija

Manipulisanje memorijom u Delphiju podleže dvama jednostavnim pravilima: morate ukloniti svaki objekat koji kreirate, i svaki objekat morate ukloniti samo jednom. Delphi podržava tri tipa manipulisanja memorijom za dinamičke elemente (to jest, elemente koji se ne nalaze na steku i globalnoj memoriji):

- Svaki put kada kreirate objekat, morate i da ga oslobodite. Ukoliko to ne učinite, memorija koju koristi objekat se neće osloboditi sve dok program ne prestane da se izvršava.
- Kada kreirate komponentu, možete navesti vlasnika komponente, prosleđujući vlasnika konstruktoru komponente. Vlasnik komponente (obično formular) postaje odgovoran za uklanjanje svih objekata koje poseduje. Drugim rečima, kada oslobodite formular, on će takođe osloboditi sve komponente koje poseduje. Dakle, ukoliko kreirate komponentu i dodelite joj vlasnika, ne morate imati na umu da je uklonite.
- Kada postavite memoriju za stringove, dinamičke nizove i objekte na koje se referiše promenljivima interfejsa (koje ćemo razmatrati na kraju ovog poglavlja), Delphi automatski oslobađa memoriju kada referenca izađe iz opsega. Nije potrebno da oslobodite string: kada postane nedostupan, njegova memorija se oslobađa.

Videćemo kako ovakvo manipulisanje memorijom utiče na primere kada budemo razmatrali aplikacije koje sadrže više formulara, u Delu II ove knjige.

Uklanjanje objekata samo jednom

Drugi problem je u tome da kada destruktor objekta pozovete dva puta, tada će se javiti greška. Destruktor (destructor) je metod kojim se oslobađa (de-allocate) memorija. Mi možemo napisati kod destruktora zaobilazeći unapred određeni destruktor `Destroy` da bismo omogućili objektu da izvrši neki kod pre nego što bude uklonjen. U Vašem kodu, naravno, ne morate rukovati oslobađanjem memorije — to je nešto što će Delphi učiniti za Vas.

Destruktor `Destroy` je jednostavno virtuelni destruktor klase `TObject`. Većina klasa koje zahtevaju kod za korisničko oslobađanje memorije kada se objekti uklanjaju, zaobilaze ovaj virtuelni metod. Razlog zbog koga nikada ne bi trebalo da definišete novi destruktor je taj što se objekti zapravo uklanjaju pozivom metoda `Free`, a ovaj metod poziva virtuelni destruktor `Destroy` (moguće verziju koja zaobilazi unapred određeni destruktor) za Vas.

Kao što sam pomenuo, `Free` je jednostavno metod klase `TObject` koji naslеđuju ostale klase. Metod `Free` u osnovi proverava da li aktuelni objekat (`Self`) nije `nil` pre nego što pozove virtuelni destruktor `Destroy`.

NAPOMENA

Možda se pitate zašto možete bez problema pozvati `Free` ukoliko je referenca objekta `nil`. Razlog je što je metod `Free` poznat na određenoj memorijskoj lokaciji, dok se virtuelna funkcija `Destroy` određuje u vreme izvršavanja prema tipu objekta, što je veoma opasna operacija ukoliko objekat više ne postoji. ■

Evo pseudokoda (stvarni kod se nalazi u RTL fajlovima izvornog koda koji je napisan u asembleru):

```
procedure TObject.Free;
begin
  if Self <> nil then
    Destroy;
end;
```

Zatim možemo obratiti pažnju na funkciju `Assigned`. Kada prosledimo pokazivač ovoj funkciji, ona samo testira da li je pokazivač `nil`, tako da su naredne dve linije ekvivalentne (iz primera CountObj2), bar u većini slučajeva:

```
if (Assigned (ToDestroy) then ...
if ToDestroy <> nil then ...
```

Primetićete da ovi iskazi testiraju samo da li pokazivač nije `nil`; ne proveravaju da li je to validan pokazivač. Ukoliko napišete sledeći kod:

```
ToDestroy.Free;
If ToDestroy <> nil then
  ToDestroy.DoSomething;
```

test će biti zadovoljen i javiće se greška u liniji u kojoj je poziv metodu objekta. Važno je shvatiti da poziv metodu `Free` ne dodeljuje objektu `nil`.

Automatsko dodeljivanje `nil` objektu nije moguće. Možete imati nekoliko referenci na isti objekat, a Delphi ih ne beleži. Istovremeno, u okviru metoda (kao što je `Free`) mi možemo operisati sa objektom, ali ne znamo ništa o referencama objekta — memorijskim adresama promenljive koju smo koristili pri pozivu metoda. Drugim rečima, unutar metoda `Free` ili bilo kog drugog metoda

klase, mi znamo memorijsku adresu objekta (`Self`), ali ne znamo memorijsku lokaciju promenljive koja se referiše na objekat, kao što je `ToDestroy`. Zbog toga metod `Free` ne može uticati na promenljivu `ToDestroy`.

Ipak, kada pozovemo spoljašnju proceduru, recimo `FreeAndNil` u Delphiju 5, procedura zna referencu objekta koja se prosleđuje kao parametar, i može sa njom operisati. Evo Delphi koda za `FreeAndNil`:

```
procedure FreeAndNil (var Obj);
var
  P: TObject;
begin
  P := TObject (Obj);
  // clear the reference before destroying the object
  TObject (Obj) := nil
  P.Free;
end;
```

Da bismo rezimirali, sledi par uputstava:

- Uvek pozovite `Free` da biste uklonili objekte umesto da pozovete destruktor Destroy.
- Upotrebite `FreeAndNil`, ili dodelite `nil` referencama objekta posle poziva `Free`, izuzev ukoliko reference izlaze iz opsega odmah posle poziva.

Prosleđivanje i kopiranje objekata

Još jedan važan element koji ćemo razmotriti je prosleđivanje objekata kao parametara ili dodeljivanje jednog objekta drugom objektu. Ukoliko napišete

```
var
  Button2: TButton;
Begin
  Button2 := Button1;
```

Vi ne kreirate novi objekat već novu referencu na isti objekat u memoriji. U memoriji postoji samo jedan objekat, a obe promenljive, `Button1` i `Button2`, se referišu na taj objekat. Isto se dešava ukoliko prosledite objekat kao parametar funkciji: Vi ne kreirate novi objekat već se na isti objekat referišete na dva različita mesta u kodu.

Na primer, ukoliko napišete ovu proceduru i pozovete je na sledeći način, promenićete natpis `Button1` ili `Button` kako želite:

```
procedure ChangeCaption (Button: TButton; Text: string);
begin
  Button.Caption := Text;
end;

// call . . .

ChangeCaption (Button1, 'Hello')
```

Šta ako je umesto toga potrebno da kreirate novu kontrolu? U osnovi je potrebno da je kreirate i kopirate svako od relevantnih svojstava. Neke klase, naročito neke VCL klase izvedene iz klase `TPersistent`, definiše metod `Assign` za kopiranje podataka objekta. Na primer, možete napisati

```
ListBox1.Items.Assign (Memo1.Lines);
```

Čak i da ta svojstva direktno dodelite, Delphi će izvršiti za Vas sličan kod. Zapravo, metod `SetItems`, povezan sa svojstvom elemenata liste, poziva metod `Assign` klase `TStringList` reprezentujući aktuelne elemente.

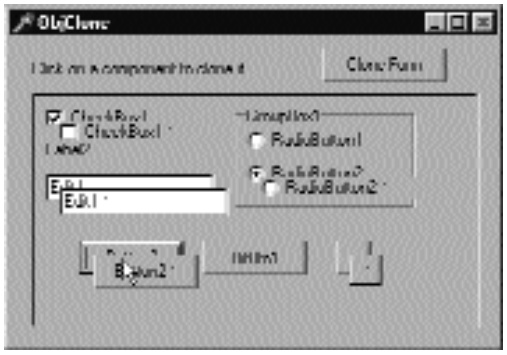
Možete upotrebiti složene tehnike usmeravanja na klon komponente u memoriji, ali u većini slučajeva kreiranje novog objekta istog tipa kao što je aktuelni objekat i dodeljivanje nekoliko svojstava obavlja posao. Da biste to učinili, možete upitati komponentu koje je klase, a zatim upotrebiti referencu klase za kreiranje novog objekta tog tipa. Evo koda (izdvojenog iz primera CountObj), kojim se klonira objekat `Sender`:

```
procedure TForm1.ClickComp (Sender: TObject);
var
  ControlText: string;
begin
    with TControlClass (Sender.ClassType).Create (Self) do
    begin
      Parent := (Sender as TControl).Parent;
      Left := (Sender as TControl).Left + 10;
      Top := (Sender as TControl).Top + 10;
      SetLength (ControlText, 50);
      (Sender as TControl).CetTextBuf(
        PChar(ControlText), 50);
      ControlText := PChar(ControlText) + ' *';
      SetTextBuf (PChar (ControlText));
    end;
end;
```

Ovaj metod uzima klasu objekta `Sender`, komponente koju korisnik klikne, i poziva konstruktor `Create`. Da bi pozvao konstruktor `Create` klase `TControl` umesto poziva konstruktora klase `TObject`, program mora da pretvori referencu klase u odgovarajući tip. Kada pretvorimo u tip `TControlClass` i zatim pozovemo `Create`, rezultat je objekat klase `TControl`. Ovaj objekat se koristi unutar iskaza `with`, a program određuje njegova svojstva `Parent`, `Left` i `Top` koristeći informacije izdvojene iz kontrole `Sender`.

Na kraju iskaza `with` program izdvaja tekst objekta `Sender` koristeći metod `GetTextBuf`, koji je na raspolaganju za svaku kontrolu. Zapravo, svojstva `Text` i `Caption` nisu definisana unutar klase `TControl`. Posle dodavanja asteriska stringu, program koristi string kao novi tekst kontrole, ponovo pozivajući metod `SetTextBuf` klase `TControl`.

Efekat kloniranja nekih kontrola možete videti na slici 3.3. Program ObjClone takođe može da klonira ceo formular, koristeći sličnu tehniku.



SLIKA 3.3 *Primer ObjClone*

Rukovanje izuzecima

Poslednja interesantna mogućnost Object Pascala koju ćemo razmotriti u ovom poglavlju je rukovanje izuzecima (exception handling). Ideja izuzetaka je da program učini robusnijim dodavanjem mogućnosti rukovanja softverskim i hardverskim greškama na jednostavan i uniforman način. Program može da preživi takve greške ili prekine bezbedno izvršavanje omogućavajući da sačuvate podatke pre prekida. Izuzeci Vam omogućavaju da odvojite kod rukovanja greškama od normalnog koda umesto da ta dva koda preplićete. Na kraju ćete pisati kod koji je kompaktniji i manje isprekidan poslovima održavanja koji nisu povezani sa stvarnom namenom programa.

Još jedna dobit je da izuzeci definišu uniformni i univerzalni mehanizam prijavljivanja grešaka, koji takođe koriste Delphi komponente. U vreme izvršavanja Delphi se poziva na izuzetke kada nešto pođe naopako. Ukoliko je Vaš kod pravilno napisan, on može prepoznati problem i pokušati da ga reši; inače, izuzetak se prosleđuje kodu poziva, i tako dalje. Na kraju, ukoliko nijedan deo Vašeg koda ne obrađuje izuzetak, obrađuje ga Delphi prikazivanjem standardne poruke o grešci i pokušava da nastavi izvršavanje programa.

Ceo mehanizam se zasniva na četiri ključne reči:

- `Try` označava početak zaštićenog bloka koda.
- `Except` označava kraj zaštićenog bloka koda i uvodi iskaze za obradu izuzetka sledećom sintaksom: on exception-type do statement

  ```
  on exception-type do stetement
  ```

- `Finally` se koristi za označavanje bloka koda koji se mora uvek izvršiti, čak i kada se desi izuzetak.
- `Raise` je iskaz kojim se generiše izuzetak. Većinu izuzetaka na koje naiđete u Delphi programiranju će generisati sistem, ali se takođe možete pozvati na izuzetak u Vašem kodu kada otkrije nepravilne ili nedosledne podatke u vreme izvršavanja. Ključna reč `raise` se može, takođe, upotrebiti unutar obrade da biste se ponovo pozvali na izuzetak, to jest, da bi se izuzetak prosledio narednoj obradi.

Evo primera jednostavnog zaštićenog bloka:

```
function DivideTwicePlusOne (A, B: Integer): Integer;
begin
  try
    // error if B equals 0
    Result := A div B;
    // do something else . . . skip if exception is raised
    Result := Result div B;
    Result := Result + 1;
  except
    on EDivByZero do
      Result := 0;
  end;
end;
```

U iskazu obrade izuzetka ulovićemo izuzetak `EDivByZero`, koji je definisan u Delphiju. Postoji veliki broj ovakvih izuzetaka koji se odnose na probleme u vreme izvršavanja (kao što je deljenje nulom ili pogrešno dinamičko pretvaranje), do raznih resursnih problema Windowsa ili grešaka komponenata (kao što je pogrešan indeks). Programeri takođe mogu definisati sopstvene izuzetke; jednostavno kreirajte novu potklasu unapred određene klase izuzetka ili neke njene potklase:

```
type
  EArrayFull = class (Exception);
```

Kada dodate novi element nizu koji je već popunjen (verovatno zbog logičke greške u programu), možete pozvati odgovarajući izuzetak kreiranjem objekta ove klase:

```
if MyArray.Full then
  raise EArrayFull.Create ('Array full');
```

Ovaj metod `Create` (nasleđen od klase `Exception`) sadrži string parametar kojim se korisniku opisuje izuzetak. Ne morate brinuti o uklanjanju objekta koji ste kreirali za ovaj izuzetak, jer će automatski biti uklonjen mehanizmom obrade izuzetaka.

Kod koji je prikazan u prethodnim isečcima je deo jednostavnog programa, nazvanog Except. Neke od rutina su zapravo malo modifikovane, kao u slučaju sledeće funckije `DivideTwicePlusOne`:

```
function DivideTwicePlusOne (A, B: Integer): Integer;
begin
  try
    // error if B equals 0
    Result := A div B;
    // do something else... skip if exception is raised
    Result := Result div B;
    Result := Result + 1;
  except
    on EDivByZero do
    begin
      Result := 0;
      MessageDlg ('Divide by zero corrected',
        mtError, [mbOK], 0);
    end;
    on E: Exception do
```

```
    begin
      Result   := 0;
      MessageDlg (E.Message,
        mtError, [mbOK], 0);
    end;
  end; // end except
end;
```

SAVET

Kada pokrenete program u debageru, debager će prekinuti izvršavanje kada naiđe na izuzetak. To je ono što normalno želite da se desi, jer ćete znati gde se izuzetak dogodio i možete videti poziv za obradu korak po korak. U slučaju test-programa Except ovo ponašanje će zbuniti program. Zapravo, iako je kod pravilno pripremljen za izvršavanje, debager će prekinuti izvršavanje kod linije izvornog koda koja je najbliža pozivu na izuzetak. Zatim, kretanjem korak po korak kroz kod možete videti kako se izuzetak obrađuje. Ukoliko želite da nastavite izvršavanje programa kada se naiđe na izuzetak, pokrenite program iz Windows Explorera, ili privremeno onemogućite opciju Stop on Delphi Exceptions na strani Language Options okvira za dijalog Debugger Option (do kojeg možete doći komandom Tools➥Debugger Options). ■

U ovom kodu postoje dve različite obrade izuzetka u okviru istog bloka `try`. Možete imati bilo koji broj ovakvih obrada koje se obrađuju sekvencijalno. Zbog toga je potrebno da najširi izuzetak (obradu predaka klasa `Exception`) smestite na kraj.

Zapravo, upotrebom hijerarhije izuzetaka, rutina za obradu se takođe poziva za potklase tog tipa, kao što bi to učinila bilo koja procedura. Ponovo imamo polimorfizam u akciji. Međutim, imajte na umu da upotreba rutina za obradu izuzetaka za svaki izuzetak, kao što je ovaj iznad, obično nije dobro rešenje. Bolje je nepoznate izuzetke ostaviti Delphiju. Unapred određena obrada izuzetka u VCL-u prikazuje poruke o grešci klase izuzetka u okviru za poruke, a zatim se nastavlja normalno izvršavanje programa. Vi, zapravo, možete izmeniti normalnu obradu izuzetka događajem `Application.OnException`, kao što je pokazano u primeru ErrorLog kasnije u ovom poglavlju.

Još jedan važan element prethodnog koda je upotreba objekta izuzetka u rutini za obradu (pogledajte on `E: Exception do`). Objekat `E` klase `Exception` dobija vrednost objekta izuzetka koji je prosleđen iskazom `raise`. Kada radite sa izuzecima, imajte na umu pravilo: pozivate se na izuzetak kreiranjem objekta i obrađujete ga naznačavanjem tipa izuzetka. Ovaj način ima važnu prednost jer, kao što smo videli, kada obrađujete tip izuzetka, Vi zapravo obrađujete izuzetke tipa koji navodite kao svaki od izvedenih tipova.

Delphi definiše hijerarhiju izuzetaka i možete odabrati različit način obrade za svaki od tipova izuzetaka ili možete obrađivati grupe izuzetaka odjednom. Spisak Delphi klasa izuzetaka možete naći na adresi `www.marcocantu.com/d5ref`.

Izuzeci i stek

Kada se program pozove na izuzetak i aktuelna rutina ne obrađuje taj izuzetak, šta se dešava sa stekom poziva funkcije? Program počinje da pretražuje funkcije koje su već na steku da bi pronašao rutinu za obradu. To znači da program napušta postojeće funkcije i ne izvršava preostale linije koda. Da biste shvatili kako ovo funkcioniše, možete koristiti debager ili dodati pozive okviru za poruke u kod, da biste bili informisani kada je određena linija koda izvršena. U narednom primeru, Except2, ja sam iskoristio drugi pristup.

Na primer, kada kliknete kontrolu Raise2 u formularu primera Except2, pozvaće se izuzetak koji neće biti obrađen, tako da se poslednji deo koda nikada neće izvršiti:

```
procedure TForm1.ButtonRaise2Click (Sender: TObject);
begin
  // ungarded call
  AddToArray (24);
  ShowMessage ('Program never gets here');
end;
```

Primetićete da ovaj metod poziva proceduru AddToArray, koja se uvek poziva na izuzetak. Kada se izuzetak obradi, tok se nastavlja posle rutine za obradu izuzetka, a ne posle koda koji se pozvao na izuzetak. Razmotrite izmenjeni metod:

```
procedure TForml.ButtonRaiselClick(Sender: TObject);
begin
  try
    // this procedure raises an exception
    AddToArray (24);
    ShowMessage ('Program never gets here');
  except
  on EArrayFull do
      ShowMessage ('Handle the exception');
  end;
  ShowMessage ('ButtonRaise1Click call completed');
end;
```

Poslednji poziv ShowMessage će biti izvršen odmah posle drugog poziva, dok se prvi poziv ignoriše. Ja Vas upućujem na to da pokrenete program, promenite njegov kod i eksperimentišete sa njim sve dok u potpunosti ne shvatite njegov tok izvršavanja kada se program pozove na izuzetak.

Blok finally

Postoji i četvrta ključna reč za obradu izuzecima koju sam pomenuo, ali je do sada nisam koristio To je ključna reč finally. Blok finally se koristi da bi se izvršile neke akcije (obično operacije čišćenja) koje je potrebno uvek izvršiti. Zapravo, iskazi u bloku finally se izvršavaju bez obzira na to da li se desio izuzetak. Običan kod posle bloka try se izvršava samo ukoliko se nije desio izuzetak ili ukoliko se desio i ukoliko je obrađen. Drugim rečima, kod u bloku finally se uvek izvršava posle koda bloka try, čak i kada se nije desio izuzetak.

Razmotrite ovaj metod (deo primera Except3), koji izvršava neke operacije koje zahtevaju dosta vremena i koji koristi kursor sa peščanim satom da bi naznačio korisniku da se nešto događa:

```
procedure TForml.BtnWrongClick(Sender: TObject);
var
  I, J: Integer;
begin
  Screen.Cursor := crHourqlass;
  J := 0;
  // long (and wrong) computation...
  for I := 1000 downto 0 do
    J := J + J div I;
  Messageolg ('Total: ' + IntToStr (J). mtInformation, [mbOK], 0);
  Screen.Cursor := crDefault;
end;
```

Pošto postoji greška u algoritmu (jer promenljiva I može dobiti vrednost 0 a koristi se i za deljenje), program će prekinuti izvršavanje, ali neće promeniti unapred određeni kursor. To je ono čemu služi blok `try-finally`:

```
procedure TForm1.BtnTryFinallyClick (Sender: TObject);
var
  J, J: Integer;
  begin
    Screen.Cursor crHourglass;
    J := 0;
    try
      // long (and wrong) computation. . .
      for I := 1000 downto 0 do
        J := J + J div I;
      MessageDlg ('Total: ' + IntloStr (J),
        mtInformation, [mbOK], 0);
    finally
      Screen.Cursor := crDefault;
    end
end;
```

Kada program izvrši ovu funkciju, uvek će resetovati kursor, bez obzira na to da li se desio izuzetak (bilo kakav) ili ne. Nedostatak ove verzije funkcije je taj da ona ne obrađuje izuzetak. Dovoljno čudno, ali to nije moguće. Iza bloka `try` se može nalaziti iskaz `except` ili `finally`, ali se ne mogu istovremeno nalaziti oba iskaza. Tipično rešenje je upotreba dva ugnežđena bloka `try`, pri čemu je unutrašnji vezan za iskaz `finally`, a spoljašnji za iskaz `except` ili obrnuto, već kako nalaže situacija. Evo koda treće kontrole primera Except3:

```
procedure TForml.BtnTryTryCliCk(Seflder Tobject);
var
  I, J: Integer;
begin
  Screen.Cursor := crHourglass;
  J := 0;
  try try
    // long (and wrong) computation...
    for I := 1000 downto 0 do
      J:= J + J div I;
    MessageDlg ('Total: ' + IntToStr (J),
      mtInformation, [mb0K], 0);
  finally
    Screen.Cursor := crDefault;
  end;
  except
    on E: EDivByZero do
    begin
      // re-raise the exception with
      raise Exception.Create ('Error in algorithm');
    end;
  end;
end;
```

Trebalo bi uvek da zaštitite blokove iskazom `finally` da biste izbegli nedostatak resursa ili memorije u slučaju poziva na izuzetak. Rukovanje izuzecima je verovatno manje važno s obzirom na to da Delphi može da izađe na kraj sa većinom izuzetaka.

Beleženje grešaka

U većini slučajeva nećete znati koja će operacija dovesti do izuzetka, a i ne možete (ili ne treba) svaki deo koda obaviti blokom `try-except`. Alternativni pristup je da prepustite Delphiju rukovanje svim izuzecima koje će proslediti Vama, a koje ćete obraditi događajem `OnException` globalnog objekta `Application`. U prethodnim verzijama Delphija mogli ste da obradite ovaj događaj pisanjem odgovarajućeg metoda koji biste ugradili u kod. Delphi 5 obezbeđuje novu komponentu ApplicationEvents koju možemo upotrebiti pri izradi ovog primera. (Više o globalnom objektu `Application` i novoj komponenti ApplicationEvents u Poglavlju 6.)

U primeru ErrorLog, ja sam glavnom formularu dodao kopiju komponente ApplicationEvents i dodao sam obradu njegovog događaja `OnException`:

```
procedure TFormLog.LogException (Sender: TObject; E: Exception);
var
  Filename: string;
  LogFile: TextFile;
begin
  // prepares log file
  Filename := ChangeFileExt (Application.Exename,  '.log');
  AssignFile (LogFile, Filename);
  if FileExists (FileName) then
    Append (LogFile) // open existing file
  else
    Rewrite (LogFile); // create a new one

  // write to the file and show error
  Writeln (LogFile, DateTimeToStr (Now) + ':' + E.Message);
  if not CheckBoxSilent.Checked then
    Application.ShowException (E);

  // close the file
  CloseFile (LogFile);
end;
```

NAPOMENA

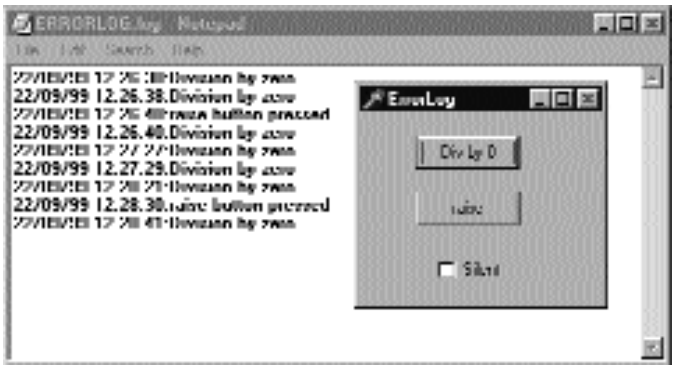
Primer ErrorLog koristi jednostavnu podršku za tekst fajlove, obezbeđenu tradicionalnim tipom podataka TextFile Turbo Pascala. Možete tekst promenljivu dodeliti fajlu, a zatim jednostavno obavljati čitanje i pisanje. Više o operacijama TextFile možete pronaći u knjizi "Osnove Pascala" (Essential Pascal) koja je dostupna na adresi `www.marcocantu.com`. ■

U globalnoj rutini za obradu izuzetaka možete zapisivati u dnevnik, na primer, datum i vreme događaja, a takođe možete odlučiti da li da izuzetak prikažete onako kako to Delphi obično čini (izvršavanjem metoda `ShowExcept` klase `TApplication`). Zapravo, Delphi po definiciji izvršava `ShowEcept` samo ukoliko ne postoji instalirana obrada izuzetka `OnException`.

Konačno, ne zaboravite da zatvorite fajl, oslobađajući bafere, svaki put kada se obradi izuzetak ili kada program prekine izvršavanje. Ja sam odabrao prvi pristup da bih izbegao da fajl ostane otvoren dok se aplikacija izvršava, i da bih otežao rad sa njim. Ovo možete postići u događaju formulara `OnDestroy`:

```
procedure TFormLog.FormDestroy (Sender: TObject);
begin
  CloseFile (LogFile);
end;
```

Formular programa sadrži polje za potvrdu da bi se odredilo njegovo ponašanje i dve kontrole kojima se generišu jednostavni izuzeci. Na slici 3.4 možete videti izvršavanje programa ErrorLog i primer dnevnika izuzetaka koji je otvoren u Notepadu.



SLIKA 3.4 *Primer ErrorLog i dnevnik koji proizvodi*

Specifikator pristupa published

Pored direktiva pristupa `public`, `protected` i `private`, možete koristiti i četvrtu direktivu, direktivu `published`. Polje ili metod `published` je dostupan ne samo u vreme izvršavanja programa već i u vreme dizajniranja. Zapravo, svaka komponenta sa Delphi Components Palette sadrži interfejs published koji koriste neki Delphi alati, naročito Object Inspector. Standardna upotreba polja `published` je važna kada pišete komponente. Obično deo komponente `published` ne sadrži polja ili metode, ali sadrži novi element jezika: svojstva.

Kada Delphi generiše formular, definicije svojih komponenata i metoda smešta u prvi deo njihove definicije, pre ključnih reči `public` i `private`. Ova polja i metodi inicijalnog dela klase su `published`. Unapred je određna direktiva `published` kada nije upotrebljena nijedna ključna reč ispred elementa klase komponente.

NAPOMENA

Da bismo bili precizniji, `published` je unapred određena ključna reč samo ukoliko je klasa kompajlirana uz pomoć direktive kompajlera `$M+` ili je izvedena iz klase koja je kompajlirana pomoću direktive kompajlera `$M+`. Budući da je ova direktiva upotrebljena u klasi `TPersistent`, većina VCL klasa i sve klase komponenata koriste po definiciji `published`. Ipak, klase koje ne sadrže komponente u Delphiju (kao što su klase `TStream` i `TList`), su kompajlirane pomoću direktive `$M-` i unapred je određeno da su javno dostupne. ■

Metodi bilo kog događaja bi trebalo da su metodi `published`, a polja koja odgovaraju Vašim komponentama formulara bi trebalo da su `published` da bi automatski bila povezana sa objektima opisanim u DFM fajlu i kreirana uz formular. Samo komponente i metodi koji se nalaze u inicijalnom `published` delu deklaracije Vašeg formulara se mogu prikazati u Object Inspectoru (u listi komponenata formulara ili u listi dostupnih metoda koja se prikazuje kada odaberete listu događaja).

Definisanje svojstava

Sada kada ste upoznali ključnu reč `published`, možemo obratiti pažnju na ostala proširenja jezika Object Pascal koja su naročito namenjena vizuelnom programiranju komponenata. Ovaj odeljak se odnosi na svojstva; kasnije ćemo se pozabaviti događajima i izraditi prvu jednostavnu komponentu.

Svojstva predstavljaju atribute koji određuju status i ponašanje objekta. Svojstvo je u osnovi naziv koji se mapira za neke metode `read` i `write` ili kojim se pristupa nekim podacima direktno. Drugim rečima, svaki put kada pročitate ili promenite vrednost svojstva, možda pristupate polju (čak i privatnom) ili pozivate metod. Na primer, evo definicije svojstva za jedan objekat podataka:

```
property Month: Integer
  read FMonth write SetMonth;
```

Da bi pristupio vrednosti svojstva `Month`, ovaj kod mora da pročita vrednost privatnog polja `FMonth`, a da bi promenio vrednost, poziva metod `SetMonth`. Moguće su različite kombinacije (na primer, mogli smo, takođe, da upotrebimo metod da bismo pročitali vrednost ili da direktno promenimo polje direktivom `write`), ali je upotreba metoda kojim se menja vrednost svojstva veoma česta. Evo nekoliko alternativa:

```
property Month: Integer
  read GetMonth write SetMonth;
property Moth: Integer
  read FMonth write FMonth;
```

SAVET

Kada napišete kod koji pristupa svojstvu, veoma je važno da shvatite da se možda poziva metod. Poenta je u tome da je za neke od metoda potrebno neko vreme za izvršavanje; takođe mogu proizvesti i brojne sporedne efekte, često uključujući (sporo) ponovno iscrtavanje komponente na ekranu. Mada su sporedni efekti obično dobro dokumentovani, potrebno je da znate da postoje, naročito kada pokušavate da optimizujete Vaš kod. ■

Direktiva svojstva `write` se takođe može izostaviti, čime svojstvo postaje svojstvo samo za čitanje (read-only). Tehnički, možete takođe izostaviti direktivu `read` i definisati svojstvo samo za pisanje (write-only), ali to nema mnogo smisla. Svojstva u vreme izvršavanja su deklarisana u

odeljku `published` definicije klase. Sve što je deklarisano u odeljku `public`, nije dostupno u *vreme dizajniranja* — samo u *vreme izvršavanja*. Sva svojstva samo za čitanje moraju biti definisana u odeljku `public` (ili u odeljcima `protected` ili `private`) jer svojstva published moraju biti za čitanje-pisanje (read-write).

Da biste pročitali ili promenili vrednost svojstva `published` u vreme izvršavanja, možete upotrebiti Object Inspector. To je alat koji vizuelno programsko okruženje Delphi obezbeđuje da bi se dozvolio pristup svojstvima. U vreme izvršavanja možete pristupiti bilo kom svojstvu `public` ili `published` čitanjem ili pisanjem njegove vrednosti.

SAVET

Imajte na umu da Object Inspector prikazuje samo svojstva komponente koja su dostupna u vreme dizajniranja, izostavljajući svojstva koja su dostupna u vreme izvršavanja. Takođe, u Delphiju 5 neka svojstva mogu da budu sakrivena, ukoliko je njihova kategorija izostavljena izdvajanjem (filtriranjem). Da biste dobili potpunu listu svojstava komponente, pogledajte Delphi Help fajlove, ne Object Inspector. ■

Da rezimiramo, uz svojstva koja su prikazana Object Inspectorom (koja su dostupna u vreme dizajniranja), postoje i druga svojstva (koja su dostupna u vreme izvršavanja), od kojih neka mogu da budu samo za čitanje (read-only). Zapamtite da obično možete dodeliti vrednost svojstvu ili vrednost možete pročitati, pa čak možete koristiti svojstva u izrazima, ali ne možete uvek proslediti svojstvo kao parametar proceduri ili metodu. To je zbog toga što svojstvo nije memorijska lokacija i ne može se koristiti kao parametar `var`; ne može se proslediti po referenci.

Nemaju sve VCL klase svojstva. Svojstva postoje u komponentama i drugim potklasama klase `TPersistent`, jer se svojstva obično mogu sačuvati u fajlu. DMF fajl, zapravo, nije ništa drugo do kolekcija svojstava published komponenata formulara. Delphi ima veliku podršku čuvanju ovakve vrste informacija, a to je naprednija tema koja se obrađuje u knjizi "Delphi priručnik za programere" *(Delphi Developers' Handbook* — Sybex, 1998).

Dodavanje svojstava formularu

Svojstva su veoma zvučan OOP mehanizam, veoma promišljena ideja enkapsulacije. U osnovi, imate naziv koji sakriva implementaciju pristupa informacijama klase (bilo direktno pristupanje podacima bilo pristupanje preko metoda). Zapravo, upotrebom svojstva ćete dobiti interfejs koji se verovatno neće menjati. Istovremeno, ukoliko samo želite da korisnicima dozvolite pristup nekim poljima klase, možete lako upakovati ta polja u svojstva umesto da ih učinite javnim. Nije potrebno pisanje dodatnog koda (kodiranje jednostavnih metoda `Set` i `Get` je veoma dosadno), a još uvek imate mogućnost promene implementacije Vaše klase. Čak i da promenite direktan pristup podacima pristupom koji koristi metod, nećete morati da promenite izvorni kod koji koristi ta svojstva. Potrebno je samo da ga ponovo kompajlirate. Razmišljajte o ovome kao o konceptu enkapsulacije koji je doveden do maksimalne snage!

Takođe, imajte na umu da je upotreba svojstava veoma laka u Code Completionu. Zbog toga, svaki put kada želite da neke informacije formulara učinite dostupnim drugom formularu, trebalo bi da upotrebite svojstvo. Jednostavno napišite u klasi formulara deklaraciju naziva svojstva i unesite:

```
property Clicks: Integer;
```

Zatim pritisnite kombinaciju tastera Ctrl+Shift+C da biste aktivirali Code Completion. Videćete sledeći efekat:

```
type
  TForrnl = class(TForm)
  private
    FClicks: Integer;
    procedure SetClicks(const Value: Integer);
  public
    property Clicks: Integer
      read FClicks write SetClicks;
  end;

implementation

procedure TForml.SetClicks(const Value: integer);
begin
  FClicks := Value;
end;
```

Ovo Vas spasava unosa, u poređenju sa prethodnim verzijama Delphija, a trebalo bi da učini upotrebu svojstava standardnom tehnikom pristupanja podacima formulara.

Po mom mišljenju, svojstva bi takođe trebalo koristiti u klasama formulara za enkapsulaciju pristupa komponentama formulara. Na primer, ukoliko imate glavni formular sa statusnom linijom koja se koristi za prikazivanje nekih informacija (i sa svojstvom `SimplePanel` sa vrednošću True), a želite da izmenite tekst iz sekundarnog formulara, možda ćete doći u iskušenje da napišete:

```
Form1.StatusBar1.SimpleText := 'new text';
```

Ovo je standardna praksa u Delphiju, ali nije dobra jer ne obezbeđuje bilo kakvu enkapsulaciju strukture formulara ili komponenata. Ukoliko imate sličan kod na mnogo mesta u aplikaciji, a kasnije odlučite da izmenite korisnički interfejs formulara (zamenivši statusnu liniju nekom drugom kontrolom ili aktiviravši više panela), moraćete da popravite kod na mnogo mesta.

Alternativa je upotreba metoda ili, što je još bolje, svojstva, da sakrijete specifičnu kontrolu. Jednostavno unesite

```
property StatusText: string
  read GetText write SetText;
```

i pritisnite ponovo kombinaciju tastera Ctrl+Shift+C, i prepustite Delphiju da doda definiciju metoda svojstva za čitanje i pisanje:

```
function TForm1.GetText: string;
begin
  Result := StatusBar1.SimpleText;
end;

procedure TForm1.SetText (const Value: string);
begin
  StatusBar1.SimpleText := Value;
end;
```

U ostalim formularima programa možete se jednostavno pozvati na svojstvo formulara `StatusText`, i ukoliko se korisnički interfejs promeni, promena će uticati samo na metode `Set` i `Get` svojstva.

Dodavanje svojstava klasi TDate

U prethodnom poglavlju smo napisali klasu `TDate`. Sada je možemo proširiti upotrebom svojstava. Ovaj novi primer, DateProp, je u osnovi proširenje primera ViewD2 iz Poglavlja 2. Evo nove deklaracije klase. Sada sadrži neke nove metode (koji se koriste za određivanje i dobijanje vrednosti svojstava) i četiri svojstva:

```
type
  TDate = class
  private
    fDate: TDateTime;
    function GetYear: Integer;
    function GetDay: Integer;
    function GetMonth: Integer;
    procedure SetDay (const Value: Integer);
    procedure SetMonth (const Value: Integer);
    procedure SetYear (const Value: Integer);
  public
    constructor Create; overload;
    constructor Create (y, m, d: Integer); overload;
    procedure SetValue (y, m, d: Integer); overload;
    procedure SetValue (NewDate: TDateTime); overload;
    function LeapYear: Boolean;
    procedure Increase (NumberOfDays: Integer = 1);
    procedure Decrease (NumberDfDays: Integer = 1);
    function GetText: string; virtual;
    property Day: Integer read GetDay write SetDay;
    property Month: Integer read GetMonth write SetMonth;
    property Year: Integer read GetYear write SetYear;
    property Text: string read GetText:
end;
```

Svojstva `Year`, `Day` i `Month` čitaju i zapisuju svoje vrednosti upotrebom odgovarajućih metoda. Evo dva koja se odnose na svojstvo `Month`:

```
function TDate.GetMonth: Integer;
var
  y, m, d: Word;
begin
  DecodeDate (fDate, y, m, d);
  Result := m;
end;

procedure TDate.SetMonth(const Value: Integer);
begin
  if (Value < 1) or (Value > 12) then
    raise EDateOutOfRange.Create ('Invalid month');
  SetValue (Year, Value, Day);
end;
```

Poziv `SetValue` izvršava određivanje vrednosti datuma, pozivajući se na izuzetak u slučaju greške. Ja sam definisao uobičajenu klasu izuzetka koja se poziva svaki put kada je vrednost van opsega:

```
type
  EDateOutOfRange = class (Exception);
```

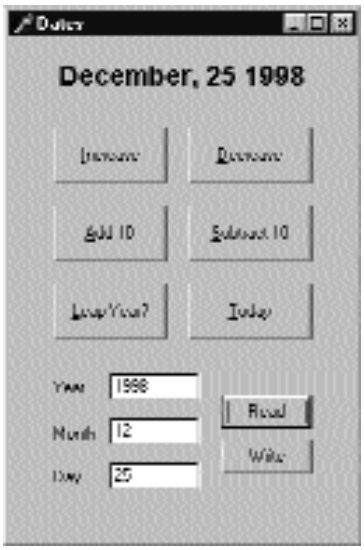
Četvrto svojstvo, `Text`, mapira samo metod za čitanje. Ova funkcija je deklarisana kao virtuelna, jer je zamenjena potklasom `TNewDate`. Ne postoji razlog zbog koga metodi `Set` i `Get` svojstva ne bi trebalo da koriste kasno povezivanje.

SAVET

Ono što je važno shvatiti u ovom primeru je da svojstva ne mapiraju podatke direktno. Oni se jednostavno proračunavaju. ■

Sada imamo klasu ažuriranu novim svojstvima te možemo ažurirati primer tako da koristi svojstva kada je to pogodno. Na primer, možemo direktno upotrebiti svojstvo `Text` i možemo upotrebiti neka polja za izmene da bismo omogućili korisniku da pročita i upiše vrednosti tri glavna svojstva (kao što možete videti na slici 3.5). Ovo se odvija kada se klikne kontrola Read:

```
procedure TDateForm.BtnReadClick (Sender: TObject);
begin
  EditYear.Text := IntToStr (TheDay.Year);
  EditMonth.Text := IntToStr (TheDay.Month);
  EditDay.Text := IntToStr (TheDay.Day);
end;
```



SLIKA 3.5 *Ažurirani formular primera DateProp u vreme izvršavanja*

Kontrola Write obavlja obrnutu operaciju. Možete napisati kod na neki od sledećih načina:

```
// direct use of properties
TheDay.Year := StrToInt (EditYear.Text);
TheDay.Month := StrToInt (EditMonth.Text);
TheDay.Day := StrToInt (EditDay.Text);
// update all values at once
TheDay.SetValue (StrToInt (EditMonth.Text),
  StrToInt (EditDay.Text),
  StrToInt (EditYear.Text);
```

Razlika između ova dva pristupa se odnosi na ono što se dešava kada unos ne odgovara legalnom datumu. Kada svaku vrednost odredimo pojedinačno, program može promeniti godinu i zatim pozvati izuzetak i preskočiti izvršavanje ostatka koda tako da se datum samo delimično promeni. Kada podesimo sve vrednosti odjednom, ili su sve ispravne i podešene, ili je jedna nepravilna i objekat datuma ostaje nepromenjen.

SAVET

Metod `SetValue` ove klase i tri svojstva imaju iste veze kao i metod `SetBounds` kakav ima klasa `TControl` prema svojstvima `Left`, `Top`, `Width` i `Height`. Zapravo, pod nekim specijalnim okolnostima isti problem, kakav je prethodno opisan, se javlja sa ovim pozicionim svojstvima kontrola. ■

Događaji u Delphiju

Kada korisnik učini nešto sa komponentom, recimo klikne komponentu, ona generiše događaj. Druge događaje generiše sistem kao odgovor na poziv metodu, ili promenu nekog od svojstava komponente (ili čak neke druge komponente). Na primer, ukoliko postavite fokus na komponentu, komponenta koja je do tada imala fokus, gubi fokus aktivirajući odgovarajući događaj.

Tehnički, većina Delphi događaja se aktivira kada pristigne odgovarajuća Windowsova poruka, iako događaji ne odgovaraju porukama u relaciji jedan na jedan. Delphi događaji su izgleda većeg nivoa od Windows poruka, a Delphi obezbeđuje veliki broj dodatnih poruka između komponenata.

Sa teorijske strane, događaj je rezultat poruke koja je poslata prozoru, a taj prozor (ili odgovarajuća komponenta) može reagovati na poruku. Sledeći ovakav pristup, da bismo obradili događaj kada se klikne kontrola, bila bi nam potrebna potklasa klase `TButton` i bilo bi potrebno da dodamo novu obradu događajem.

U praksi je kreiranje nove klase previše složeno da bi to bilo razumno rešenje. U Delphiju, obrada događaja komponente je obično metod formulara koji sadrži komponentu, ne sama komponenta. Drugim rečima, komponenta se oslanja na svog vlasnika, formular, pri obradi njenih događaja. Ova tehnika se naziva autorizacija (delegation) i predstavlja osnovu Delphi-jevog modela komponenata.

Događaji su svojstva

Još jedan važan koncept je da su događaji svojstva. To znači, da biste obradili događaj komponente, Vi dodeljujete odgovarajuće svojstvo događaja, kao što smo to uradili ranije u primeru CountObj2 ovog poglavlja. Kada dva puta kliknete događaj u Object Inspectoru, dodaje se novi metod vlasniku formulara i dodeljuje se odgovarajućem događaju svojstva komponente.

Eto zbog čega je moguće da nekoliko događaja dele istu obradu događaja ili da promene obradu događaja u vreme izvršavanja. Da biste koristili ovu mogućnost, nije potrebno veliko znanje o jeziku. Zapravo, kada odaberete događaj u Object Inspectoru, možete kliknuti kontrolu sa strelicom na desnoj strani događaja da biste videli listu kompatibilnih metoda — listu metoda koji imaju isti tip pokazivača metoda. Upotrebom Object Inspectora lako je odabrati isti metod za isti događaj neke druge komponente ili za različite, kompatibilne događaje iste komponente.

Dodavanje događaja klasi TDate

Kao što smo dodali svojstva klasi `TDate`, možemo dodati i događaj. Događaj će biti veoma jednostavan. Nazvaćemo ga `OnChange` i može se upotrebiti da upozori korisnika komponente da je vrednost datuma promenjena. Da bismo definisali događaj, definisaćemo svojstvo koje mu odgovara i dodaćemo neke podatke da bismo sačuvali stvarni pokazivač metoda na koji se referiše događaj. Ovo su nove definicije koje su dodate klasi:

```
type
  TDate = class
  private
    FOnChange: TNotifyEvent;
    ...
  protected
    procedure DoChange; dynamic;
    ...
  public
    property OnChange: TNotifyEvent
      read FOnCnahge write FonChange;
    ...
  end;
```

Definicija svojstva je, zapravo, veoma jednostavna. Korisnik ove klase može dodati novu vrednost svojstvu i, zbog toga, privatnom polju `FOnChange`. Klasa ne dodeljuje vrednost ovom polju `FOnChange`. Korisnik komponente je onaj koji dodeljuje vrednost. Klasa `TDate` jednostavno poziva metod koji je sačuvan u polju `FOnChange` kada se menja vrednost datuma. Naravno, poziv se izvršava samo ukoliko je dodeljeno svojstvo događaja. Metod `DoChange` (deklarisan kao dinamički metod jer je to uobičajeno kada događaj poziva metod) obavlja testiranje i poziv metodu:

```
procedure TDate.DoChange;
begin
  if Assigned (FOnChange) then
    FOnChange (Self);
end;
```

Metod `DoChange` se poziva svaki put kada se promeni jedna od vrednosti, kao što je to slučaj u sledećem metodu:

```
procedure TDate.SetValue (y, m, d: Integer);
begin
  fDate := EncodeDate (y, m, d);
  // fire the event
  DoChange;
```

Ukoliko sada pogledamo program koji koristi ovu klasu, možemo značajno pojednostaviti njegov kod. Prvo, dodaćemo novi metod klasi formulara:

```
type
  TDateForm = class (TForm)
    . . .
  procedure DateChange (Sender: TObject);
```

Kod ovog metoda jednostavno ažurira oznaku aktuelnom vrednošću svojstva Text objekta `TDate`:

```
procedure TDateForm.DateChange;
begin
  LabelDate.Caption := TheDay.Text;
end;
```

Ova obrada događaja se zatim instalira u metod `FormCreate`:

```
procedure TDateForm.FormCreate (Sender: TObject);
begin
  TheDay := TDate.Init (7, 4, 1997);
  LabelDate.Caption := TheDay.Text;
  // assign the event handler for future changes
  TheDay.OnChange := DateChange;
end;
```

Ovo izgleda kao mnogo posla. Jesam li lagao kada sam rekao da će nas obrada događaja osloboditi malo kodiranja? Ne. Sada, kada smo dodali nešto koda, možemo potpuno zaboraviti na ažuriranje oznake kada se promene podaci objekta. Ovde je, kao primer, data obrada događaja `OnClick` jedne od kontrola:

```
procedure TDateForm.BtnIncreaseClick (Sender: TObject);
begin
  TheDay.Increase;
end;
```

Isti pojednostavljeni kod je prisutan u mnogim rutinama za obradu događaja. Kada jednom instaliramo obradu događaja, ne moramo pamtiti da je potrebno ažurirati oznaku. Ovim se eliminiše značajan potencijalni izvor grešaka u programu. Primetićete da smo morali da napišemo kod na početku, jer to nije komponenta koja je instalirana u Delphiju već samo klasa. Kada je u pitanju komponenta, jednostavno ćete odabrati rukovanje događajem u Object Inspectoru i napisati jednu liniju koda da biste ažurirali komponentu. To je sve. Koliko je teško napisati novu komponentu u Delphiju? To je zapravo toliko jednostavno, da ću Vam ja to pokazati u narednom odeljku.

NAPOMENA

Bilo je predviđeno da ovo bude kratak uvod u ulogu svojstava i događaja i u pisanje komponente. Razumevanje ovih mogućnosti je veoma važno za svakog Delphi programera. Ukoliko je Vaš cilj da napišete složene nove komponente, naći ćete mnogo više informacija o svim ovim temama u Poglavlju 13. ■

Kreiranje komponente TDate

Sledeći korak, koji je veoma jednostavan, je da našu klasu `TDate` pretvorimo u komponentu. Prvo moramo našu klasu izvesti iz klase `TComponent`, umesto iz unapred određene klase `TObject`. Evo koda:

```
type
  TDate = class (TComponent)
    . . .
  public
  constructor Create (AOwner: TComponent); overload; override;
  constructor Create (y, m, d: Integer); reintroduce; override;
```

Kao što možete videti, drugi korak je bio dodavanje novog konstruktora klasi zaobilazeći unapred određeni konstruktor komponenata, da bismo obezbedili pogodnu inicijalnu vrednost. Pošto postoji preopterećena verzija, takođe je bilo potrebno upotrebiti direktivu `reintroduce` da bismo izbegli da kompajler proizvede poruku upozorenja. Kod novog konstruktora jednostavno podešava datum na današnji datum, posle poziva konstruktora osnovne klase:

```
constructor TDate.Create (AOwner: TComponent);
var
  Y, D, M: Word;
begin
  inhereted Create (AOwner);
  // today . . .
  fDate := Date;
```

Kada smo to obavili, potrebno je da klasi dodamo jedinicu koja definiše našu klasu (fajl `DATES.PAS` u direktorijumu `DATECOMP`), proceduru `Register`. (Postarajte se da reč počinje velikim slovom R, inače neće biti prepoznata.) Ovo je neophodno da bismo dodali komponentu Delphijevoj Component Palette. Jednostavno deklarišite proceduru kojoj nisu potrebni parametri, u delu jedinice `interface`, a zatim napišite ovaj kod u odeljku `implementation`:

```
procedure Register;
begin
  RegisterComponents ('Md1', [TDate]);
end;
```

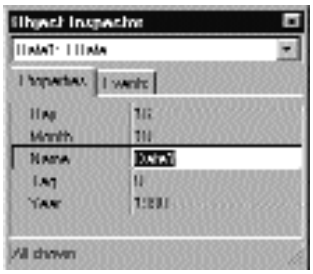
Ovaj kod dodaje novu komponentu strani *Md5* Components Palette, kreirajući stranu ukoliko je to neophodno. Usput, to je strana koju ću koristiti za sve komponente koje ćemo izraditi u ovoj knjizi.

Poslednji korak je instaliranje komponente. Za ovaj jednostavan primer nećemo kreirati novi paket. Umesto toga možemo instalirati komponentu u unapred određeni paket Borland User Component (fajl nazvan `DCLUSR50.DPK` koji se čuva u `LIB` direktorijumu Delphija). U Poglavlju 13 ćemo videti kako da napravimo nove pakete.

Da biste učinili da komponenta bude na raspolaganju, odaberite element menija Component➥Install Component, odaberite stranu *Into existing package* (u postojeći paket — ovo bi trebalo da bude unapred određeno), odaberite naziv fajla `DCLUSR50.DPK` paketa (ponovo, ovo je unaperd određeno ukoliko nikada niste instalirali komponente) i unesite naziv fajla jedinice komponente, `DATES.PAS`. Sada jednostavno kliknite OK, a Delphi će ažurirati paket, kompajlirati ga i pitati da li da ga instalira u Delphi (ukoliko to već niste učinili).

Ukoliko sada pređete na Components Palette, trebalo bi da sadrži novu stranu *Md* na kojoj je nova komponenta. Komponenta će biti predstavljena unapred određenom ikonom za Delphi komponente. Sada možete postaviti komponentu na formular nove aplikacije i početi da manipulišete njenim svojstvima u Object Inspectoru, što možete videti na slici 3.6. Takođe, možete obraditi događaj `OnChange` mnogo lakše nego u prethodnom primeru.

Pored pokušaja da izradite Vaš primer aplikacije u kojoj se koristi ova komponenta (nešto što Vam ja preporučujem), možete otvoriti primer DateComp, koji je ažurirana verzija komponente koju smo izradili korak po korak u prethodnih nekoliko odeljaka ovog poglavlja. To je, u osnovi, pojednostavljena verzija primera DateEvt jer je sada obrada događaja direktno dostupna u Object Inspectoru.



SLIKA 3.6 *Svojstva naše nove komponente TDate u Object Inspectoru*

UPOZORENJE

Ukoliko otvorite primer DateComp pre nego što instalirate novu komponentu, Delphi neće prepoznati komponentu kada otvori formular i prikazaće poruku o grešci. Nećete moći da kompajlirate program ili učiniti da proradi, sve dok ne instalirate novu komponentu. ■

Upotreba interfejsa

Nasuprot onome što se dešava u C++, Delphi model nasleđivanja ne podržava višestruko nasleđivanje. To znači da svaka klasa može imati jednu osnovnu klasu. Upotrebna vrednost višestrukog nasleđivanja je tema za raspravu. Odsustvo ove konstrukcije u Delphiju se može posmatrati i kao nedostatak (jer gubite deo moći C++) i kao prednost (jer dobijate jednostavniji jezik i manje problema). Moje mišljenje je da Delphi interfejs obezbeđuje fleksibilnost i snagu deklarisanja podrške za više interfejsa koji su implementirani u jednoj klasi, pri čemu se izbegavaju problemi nasleđivanja višestrukih implementacija. Umesto da budem uvučen u raspravu, ja ću jednostavno pretpostaviti da je korisno tretirati jedan objekat iz više "perspektiva", smatrati ga generičkim objektom različitih osnovnih klasa. Ali, pre nego što izradim primer koji sledi ovaj princip, potrebno je da predstavim ulogu interfejsa u Object Pascalu.

NAPOMENA

Tehnike koje su obrađene u ovom poglavlju se takođe koriste za implementaciju COM objekata, i ja ću ih detaljnije obraditi u Poglavlju 15. Za sada, posmatrajmo ih jednostavno kao elemente jezika. ■

Deklarisanje interfejsa

Pored deklarisanja apstraktnih klasa (klasa sa apstraktnim metodima), u Delphiju takođe možete napisati čisto apstraktnu klasu (purely abstract class), to jest, tip klase koji sadrži samo apstraktne metode. To se postiže upotrebom specifične ključne reči kao što je `interface`. Zbog toga ove klase nazivamo interfejsima (interfaces). Zapravo, interfejs nije klasa, mada može da podseća na klasu. Interfejsi nisu klase jer se smatraju potpuno odvojenim elementom, koji ima svoj osnovni interfejs, `IUnknown`, čija je uloga jednaka ulozi klase `TObject` za klase.

Borland je interfejse predstavio u Delphiju 3 uz podršku COM programiranju. Ukoliko je sintaksa jezika interfejsa kreirana za podršku COM, interfejsima nije potreban COM. Možete upotrebiti interfejse da biste implementirali apstraktne slojeve u okviru Vaše aplikacije, a da ne morate da izradite COM server objekte. Na primer, Delphi IDE prilično koristi interfejse u svojoj internoj arhitekturi. Uopšte, interfejsi imaju neke velike prednosti koje mogu da budu korisne za različite tipove programiranja:

- Klasa se može izvesti iz jedne osnovne klase, ali takođe može implementirati više interfejsa. Nedostatak je u tome što kada klasa implementira interfejs, mora obezbediti implementaciju za svaki od metoda interfejsa.
- Objekti tipa interfejs se prebrojavaju po referenci i automatski se uklanjaju kada više nema referenci na objekat. Ovaj mehanizam je sličan Delphijevom mehanizmu za dugačke stringove, a čini manipulisanje memorijom gotovo automatskim.
- VCL već obezbeđuje nekoliko osnovnih klasa da bi implementirao osnovno ponašanje koje zahteva interfejs `IUnknown`. Najjednostavnija je klasa `TInterfacedObject`.

NAPOMENA

Sa uopštenije tačke gledišta, interfejsi podržavaju nešto drugačiji model objektno orijentisanog programiranja nego što to čine klase. Objekti koji implementiraju interfejse su subjekat polimorfizma za svaki od interfejsa koji podržavaju. Model interfejs je zaista moćan. Ali, kada sam to rekao, nisam imao nameru da određujem koji pristup je bolji. Očigledno, interfejsi više koriste enkapsulaciju i obezbeđuju slobodnije povezivanje između klasa nego što to čini nasleđivanje. ■

Evo sintakse deklaracije interfejsa (koja, po konvenciji, počinje velikim slovom I):

```
type
  ICanFly = interface
    ['{10000000-0000-0000-0000-000000000000}']
  function Fly: string;
  end;
```

NAPOMENA

Da bi pravilno funkcionisao, svaki interfejs mora imati predhodni numerički ID. Teorijski gledano to bi trebalo da budu jedinstveni GUIDi, koje generiše Delphi editor kada pritisnete kombinaciju tastera Ctrl+Shift+G, ali ukoliko ne planirate da izvezete ove objekte poslužiće bilo koji broj (više o GUIDima u Poglavlju 15). Ovi GUIDi su neophodni čak i ukoliko ne planirate da izvezete ove klase, jer ih koristi kompajler za proveru tipa interfejsa umesto običnih interfejsa i naziva klasa. ■

Kada ste deklarisali interfejs, možete definisati klasu da biste ga implementirali:

```
type
  TAirplane = class (TInterfacedObject, ICanFly)
    function Fly: string virtual;
  end;
```

Kao što sam pomenuo, ovu klasu možete izvesti iz klase `TInheretedObject` da bi nasledila metode `IUnknown`. Mada nije obavezno implementirati metode interfejsa virtuelnim metodima, ovo je dobar pristup ukoliko želite da imate mogućnost da izmenite ove metode u budućim potklasama. Alternativna tehnika je da ponovo deklarišete interfejs tip u izvedenim klasama i ponovo povežete interfejs metode u statičke metode deklarisane u klasi.

Sada kada smo definisali implementaciju interfejsa, kao i obično možemo napisati:

```
var
  Airplane1: TAirplane;
begin
  Airplane1 := TAirplane.Create;
  Airplane1.Fly;
  Airplane1.Free;
end;
```

Ali, takođe možemo upotrebiti interfejs tip promenljive:

```
var
  Flyer1: ICanFly;
begin
  Flyer1 := TAirplane.Create;
  Flyer1.Fly;
end;
```

Čim dodelite objekat interfejs promenljivoj, Delphi automatski proverava da li obejkat implementira interfejs, koristeći specijalnu verziju operatora `as`. Možete eksplicitno izraziti ovu operaciju na sledeći način:

```
Flyer1 := TAirplane.Create as ICanFly;
```

Bilo da koristimo direktno dodeljivanje ili iskaz `as`, Delphi će obaviti jednu dodatnu stvar: pozvaće metod `_AddRef` objekta, uvećavajući brojač reference. Istovremeno, čim promenljiva `Flyer1` izađe iz objekta, Delphi će pozvati metod `_Release`, koji umanjuje brojač reference, proverava da li je brojač reference nula i, ukoliko je potrebno, uklanja objekat. Zbog toga u prethodnom listingu nema koda kojim se oslobađa objekat koji smo kreirali.

Drugim rečima, Delphi objekti na koje se referiše interfejs promenljivima, se prebrojavaju referencama, i automatski se oslobađaju kada se na njih ne referiše nijedna interfejs promenljiva.

UPOZORENJE

Kada koristite interfejs objekte, trebalo bi da im pristupate preko objektnih promenljivih ili samo preko interfejs promenljivih. Mešanje ova dva pristupa prekida šemu brojanja referenci koju obezbeđuje Delphi i možete proizvesti memorijske greške koje je veoma teško pratiti. U praksi, ukoliko ste odlučili da koristite interfejse, verovatno bi trebalo da koristite isključivo interfejs promenljive. ■

Svojstva interfejsa, autorizacija, ponovne definicije

Da bih demonstrirao nekoliko tehničkih elemenata vezanih za interfejse, ja sam napisao primer IntfDemo. Ovaj primer se zasniva na dva različita interfejsa, `IWalker` i `IJumper`, koja su definisana na sledeći način:

```
IWalker = interface
  ['(0876F200-AAD3-11D2-8551-CCA30C584521] ']
  function Walk: string;
  function Run: string;
  procedure SetPos (Value: Integer);
  function GetPos: Integer;

  property Position: Integer
    read GetPos write SetPos;
end;

IJumper = interface
  ['(0876F201-AAD3-11D2-8551-CCA30C584521)']
  function Jump: string;
  function Walk: string;
  procedure SetPos (Value: Integer);
  function GetPos: Integer;

  property Position: Integer
    read GetPos write SetPos;
end;
```

Primetićete da prvi interfejs takođe definiše svojstvo. Interfejs svojstvo je samo naziv koji se mapira za metode `read` i `write`. Ne možete mapirati interfejs svojstvo za polje, jer jednostavno interfejs ne može imati polje:

```
TRunner = class (TInterfacedObject, IWalker)
private
  Pos: Integer;
public
  function Walk: string;
  function Run: string;
  procedure SetPos (Value: Integer);
  function GetPos: Integer;
end;
```

Kod je trivijalan tako da ću ga ja preskočiti (možete ga pronaći u primeru IntfDemo). Na sličan način sam definisao klasu implementirajući interfejs `IJumper`:

```
TJumperImpl = class (TInterfacedObject, IJumper)
private
  Pos: Integer;
public
  function Jump: string;
  function Walk: string;
  procedure SetPos (Value: Integer);
  function GetPos: Integer;
end;
```

Mada se ova klasa ne razlikuje od prethodne, ja ću je koristiti na drugačiji način. U narednoj klasi, `TMyJumper`, ja ne želim da ponovim implementaciju interfejsa `IJumper` sa sličnim metodima. Umesto toga želim da autorizujem implementaciju tog interfejsa klasi koja ga već implementira. Ovo se ne može postići nasleđivanjem (ne možemo imati dve osnovne klase); umesto toga možete koristiti specifične funkcije autorizacije interfejsa jezika:

```
TMyJumper = class (TInterfacedObject, IJumper)
private
  fJumplmpl : IJumper;
public
  constructor Create;
  property Jumper: IJumper
    read fJumpImpl implements IJumper;
end;
```

Ova deklaracija naznačava da je interfejs `TJumper` implementiran za klasu `TMyJumper` poljem fJumpImpl. Ovo polje, naravno, mora zapravo implementirati sve metode interfejsa. Da bi ovo proradilo, potrebno je da kreirate odgovarajući objekat za polje kada se kreira objekat `TMyJumper`:

```
constructor TMyJumper.Create;
begin
  fJumpImpl := TJumperImpl.Create;
end;
```

Ovaj primer je jednostavan, ali uopšte, stvari se komplikuju kada počnete da menjate neki od metoda ili da dodajete druge metode koji još uvek operišu sa podacima internog objekta `fJumpImpl`. Poslednji korak je demonstriran, uz ostale mogućnosti, klasom `TAthlete`, koja implementira oba interfejsa, interfejse `IWalker` i `IJumper`:

```
TAthlete = class (TInterfacedObject, IWalker, IJumper)
private
  fJumpImpl: TJumperImpl;
public
  constructor Create;
  function Run: string; virtual;
  function Walk1: string; virtual;
  function IWalker.Walk = Walk1;
  procedure SetPos (Value: Integer);
  function GetPos: Integer;

  property Jumper: TJumperImpl
    read fJumpImpl implements IJumper;
end;
```

Jedan od interfejsa je implementiran direktno, dok je drugi autorizovan za interni objekat `fJumpImpl`. Takođe ćete primetiti da implementiranjem dva interfejsa, koja imaju zajednički metod, dolazimo do problema sa jednakim nazivima. Rešenje je u tome da preimenujemo jedan od metoda sledećim iskazom:

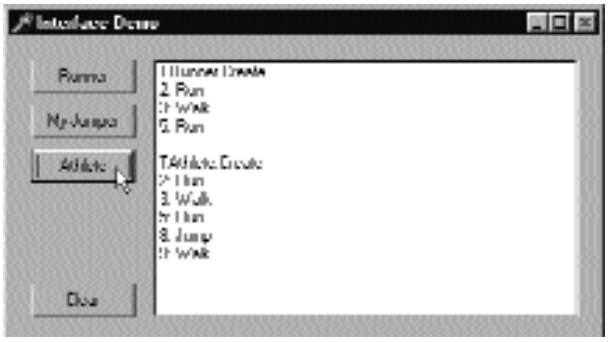
```
function IWalker.Walk = Walk1;
```

Ova deklaracija naznačava da klasa implementira metod `Walk` interfejsa `IWalker` metodom nazvanim `Walk1` (umesto metodom koji ima identičan naziv). Konačno, u implementaciji svih metoda klase je potrebno da se referišemo na svojstvo `Position` internog objekta `fJumpImpl`.

Deklarisanjem nove implementacije za svojstvo `Position`, dobićemo dve pozicije za jednog atletu (TAthlete), što je prilično čudno rešenje. Evo nekoliko primera:

```
function TAthlete.GetPos: Integer;
begin
  Result := fJumpImpl.Position;
end;

function TAthlete.Run: string;
begin
  fJumpImpl.Position := fJumpImpl.Position + 2;
  Result := IntToStr (fJumperImpl.Position) + ':Run';
end;
```

Možete dalje eksperimentisati primerom IntfDemo, koji sadrži jednostavan formular sa četiri kontrole za kreiranje i koji poziva metode različitih objekata. Ništa glamurozno, kao što možete videti na slici 3.7. Jednostavno imajte na umu da svaki poziv daje poziciju posle zahtevanog pomeranja i opis samog pomeranja.



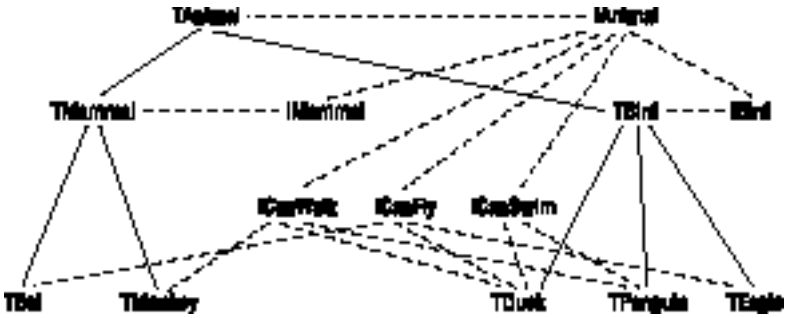
SLIKA 3.7 *Primer IntfDemo*

Primer višestrukog nasleđivanja

Posle ovog primera ćete mi dozvoliti da pređem na seriju složenijih interfejsa. Pretpostavimo da imate hijerarhiju klasa koje se odnose na životinje. Hijerarhiju možete zasnovati na standardnoj klasifikaciji (sa kategorijama kao što su sisari, ptice, insekti i tako dalje), ili ih možete kategorizovati prema sposobnostima (životinje koje lete, četvoronožne ili dvonožne životinje, mesožderi i tako dalje).

Ne postoji lak način za izražavanje ovako složene strukture jednosmernim naleđivanjem. Možete upotrebiti višestruko nasleđivanje ukoliko jezik koji koristite podržava ovu mogućnost, ili ukoliko koristite interfejse. To je ono što sam učinio za moj primer, koji predstavlja uobičajenu studiju višestrukog nasleđivanja. Ovaj program, nazvan MultInh, sadrži i hijerarhiju klasa (koja predstavlja standardnu zoološku klasifikaciju) i hijerarhiju interfejsa (koja izražava mogućnosti).

I hijerarhija klasa i hijerarhija interfejsa koriste jednostruko nasleđivanje. Samo kada pogledate kako klase implementiraju različite interfejse, možete videti da se dve hijerarhije spajaju, kao što je pokazano slikom 3.8.



SLIKA 3.8 *Složene zavisnosti između klasa i interfejsa primera MultInh*

Deklaracija ovih interfejsa i njihovih metoda je prilično dugačka, te sam odlučio da ih preskočim. Svaka od njih ima specifičan GUID i definiše jednu ili više funkcija koje kao rezultat daju string. Stvarne klase implementiraju jedan ili više od ovih interfejsa, što je predstavljeno slikom 3.8. Evo nekoliko deklaracija:

```
type
  TBird = class (TAnimal, IBird)
    function LayEggs: string; virtual;
  end;

  TEagle = class (TBird, ICanFly)
    function Kind: string; override;
    function Fly: string; virtual;
end;

TPenguin = class (TBird, ICanWalk, ICanSwim)
  function Kind: string; override;
  function Walk: string; virtual;
  function Swim: string; virtual;
```

Sada kada smo dizajnirali ovaj interfejs, kako da ga upotrebimo? Kako da kreiramo objekte ovih klasa, i kako možemo da upotrebimo polimorfizam u klasama koje implementiraju višestruke interfejse?

Polimorfizam interfejsa

Da bih upotrebio polimorfizam kod interfejsa, ja sam deklarisao i popunio niz unutar formulara programa:

```
private
  AnimIntf: array [1..5] of IAnimal;
```

Program izdvaja interfejs `IAnimal` iz novokreiranih objekata i inicijalizuje niz. To Delphi automatski obavlja kada napišete:

```
AnimIntf[1] := TEagle.Create;
```

što odgovara sledećem

```
AnimIntf[1] := TEagle.Create as IAnimal;
```

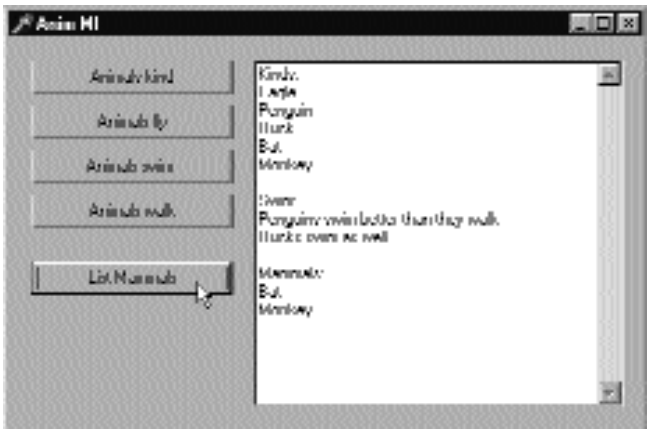
Pozivanje metoda opisanih u interfejsu `TAnimal` je pravolinijsko:

```
for I := 1 to 5 do
  Memo1.Lines.Add (AnimIntf[I].Kind);
```

Ovaj kod se zapravo izvršava kada kliknete prvu kontrolu glavnog formulara primera MultInh, kao što možete videti na slici 3.9.

Da bismo operisali sa metodima koje obezbeđuje interfejs, moramo prvo proveriti da li neki od datih objekata podržava interfejs. Pošto ne postoji operator `is` za interfejse, mi to možemo obaviti pozivajući metod `QueryInterface`:

```
var
  Fly1: ICanFly;
begin
  AnimIntf[i].QueryInterface (ICanFly, Fly1);
  if Assigned (Fly1) then
    Memo1.Lines.Add (Fly1.Fly);
```



SLIKA 3.9 *Jednostavan korisnički interfejs primera MultInh*

`QueryInterface` zahteva kao parametre promenljive za vrednost koju vraća i tip interfejsa koji treba proveriti. Budući da takođe vraća i kod greške, možemo proveriti i taj kod, kao što sam ja to uradio u jednom drugom slučaju:

```
var
  Swim1: ICanSwim;
begin
  if AnimIntf[i].QueryInterface (
    ICanSwim, Swim1) <> E_NoInterface then
  Memo1.Lines.Add (Swim1.Swim);
```

Možemo, takođe, upotrebiti iskaz `as` koristeći blok `try-except`, ali to nije rešenje koje se meni zaista dopada. (To zapravo treba da proverite u izvornom kodu programa.)

Da li je ovo višestruko nasleđivanje?

Poslednja dva fragmenta koda pokazuju da možemo upotrebiti objekat i pridružiti ga višestrukim interfejsima koje podržava. Drugim rečima, možemo smatrati patku za životinju koja pliva ili za životinju koja leti i pozvati metode oba interfejsa za jedan objekat. Možemo pridružiti objekat dvama različitim osnovnim tipovima, tako da ovo zaista nalikuje višestrukom nasleđivanju.

Ono što ne dobijamo je nasleđivanje stvarne implementacije metoda; ne postoji kod u interfejsu `IcanFly`, i kada bi postojao bilo koji kod koji dele "leteći" objekti, bilo bi potrebno da ga ponovo implementiramo za svaku klasu koja podržava ovaj interfejs. Ipak, već znamo da je moguća implementacija interfejsa u velikom broju klasa, kao što sam ja to učinio u prethodnom primeru.

Kao što sam ranije pomenuo, Borland je dodao interfejse Delphiju da bi podržao Microsoftov COM, ali se oni mogu upotrebiti kao dodatna mogućnost jezika. Najveći nedostatak je to što interfejsi moraju imati ID čak i za interne objekte, jer provera tipa interfejsa zavisi od tog broja. Drugi manji problem je u tome da ne postoji operator `is` kojim se proverava da li objekat podržava dati interfejs, ali smo videli da je veoma jednostavno imitirati ovakvo ponašanje pozivanjem `QueryInterface` jednim pozivom metoda.

Rezimirajući, da li zaista ima smisla koristiti interfejs tipove i promenljive u programu za koji nije potrebna podrška za COM? Ukoliko je program dizajniran nad složenom hijerarhijom kojoj višestruko nasleđivanje može doneti značajne pogodnosti, tada je odgovor da. Imajući na umu dodatnu složenost ovakvog dizajna, ipak se ne morate složiti.

Šta je sledeće?

Čitajući ovo poglavlje možda ste pomislili da sam objašnjavao brojne teme koje nisu povezane. To je samo delimično tačno. Reference klase, pokazivači metoda, svojstva, događaji, ključna reč `published` i izuzeci su sve mogućnosti jezika na kojima je izgrađen Delphi Visual Component Library. Ostale teme, kao što su metodi klase i interfejsi, su značajni dodaci jeziku sa kojima se svaki Delphi programer bar mora upoznati.

Pošto smo obradili osnove OOP-a u prethodnom poglavlju i sva ova proširenja u ovom poglavlju, sada možemo obratiti pažnju na strukturu VCL-a u narednom poglavlju.

Mi smo u ovom poglavlju načinili jedan dodatni korak: izradili smo prvu jednostavnu komponentu i instalirali smo je u Delphi okruženje. Ovo već demonstrira činjenicu da je komponenta zapravo klasa Object Pascala koja nasleđuje od specifične osnovne klase, klase `TComponent`. *Delphi komponente su klase*: ova naizgled jednostavna rečenica opisuje prirodu Delphi modela programiranja, ističući razlike u odnosu na alate kao što su Visual C++ ili Visual Basic. Jedini drugi programski jezik koji je blizak Object Pascalu u terminima razvoja komponenata je Java.

POGLAVLJE

4

VCL tehnike programiranja

Da bi se pojednostavio posao programiranja, Delphi obezbeđuje mnogo funkcija i klasa, moćnih i odmah spremnih za upotrebu. To uključuje, na primer, brojne standardne rutine. (Help fajlovi više ne sadrže potpunu listu ovih rutina, ali takav spisak možete pronaći na mom web sajtu na adresi www.marcocantu.com.) Još je veći i važniji Delphijev skup klasa. Neke od njih su klase komponenata, koje se prikazuju u Components Paletti, dok su ostale više opšte namene. Ovo poglavlje je okrenuto strukturi Delphi biblioteke klasa — poznatoj kao Visual Component Library (biblioteka vizuelnih komponenata — VCL), mada ne sadrži samo komponente — i daje pregled nekih klasa opšte namene.

Ukoliko jednostavno želite da upotrebite postojeće komponente i ne marite mnogo za ulaz i izlaz VCL-a, za sada možete preskočiti ovo poglavlje i preći na Deo II, koji se bavi upotrebom komponenata i ostalih klasa namenjenih Windowsu, ili Deo III, koji se odnosi na programiranje baza podataka (uključujući standardne komponente podataka). Ne zaboravite da se vratite na ovo poglavlje kada budete bili spremni da proširite svoje programersko poznavanje Delphija.

Klasa TObject

Suštinu Delphija predstavlja hijerarhija klasa. Svaka klasa sistema je potklasa klase `TObject`, tako da cela hijerarhija ima samo jedan koren. Ovo Vam omogućava da koristite tip podataka `TObject` kao zamenu za tip podataka bilo kog tipa klase sistema.

Na primer, obrada greške obično sadrži `Sender` parametar tipa `TObject`. To jednostavno znači da objekat `Sender` može biti bilo koje klase, s obzirom na to da je, na kraju krajeva, svaka klasa izvedena iz klase `TObject`. Tipičan nedostatak ovakvog pristupa je u tome što je — da biste radili nad objektom — potrebno da znate njegov tip podataka. Zapravo, kada imate promenljivu ili parametar tipa `TObject` nad njim, možete upotrebiti samo metode i svojstva definisana klasom `TObject`. Ukoliko se desi da se ta promenljiva ili parametar odnose na objekat tipa `TButton`, na primer, onda ne možete da se direktno referišete na svojstvo `Caption`. Rešenje problema možemo pronaći u činjenici da svaki objekat "zna" svoju pravu klasu, te onda možete pristupiti ovoj informaciji preko metoda `ClassType` i `ClassName`. Na primer, `ClassName` kao rezultat daje string sa nazivom klase. Pošto je ovo metod klase, možete ga primeniti i nad klasom i nad objektom. Pretpostavimo da ste definisali klasu `TButton` i obejkat te klase `Button1`. Naredni iskazi daju jednak efekat:

```
Text := Button1.ClassName;
Text := TButton.ClassName;
```

Postoje slučajevi kada je potrebno da koristite naziv klase, ali takođe može biti korisno da dobijete referencu klase na samu klasu ili na njenu osnovnu klasu. Referenca klase Vam, zapravo, omogućava da operišete sa klasom u vreme izvršavanja (kao što smo mogli da vidimo u prethodnom poglavlju), dok je naziv klase samo string. Uz pomoć metoda `ClassType` i `ClassParent` možemo dobiti ove reference klase. Kada dobijete referencu klase, možete je koristiti kao da je objekat — na primer, da biste pozvali metod `ClassName`.

Još jedan metod koji bi mogao da bude koristan je metod `InstanceSize`, koji kao rezultat daje veličinu objekta u vreme izvršavanja. (Mada ste, možda, pomislili da ovu informaciju možete dobiti globalnom funkcijom `SizeOf`, ova funkcija zapravo kao rezultat daje veličinu reference objekta — pokazivač koji je uvek veličine četiri bajta — umesto veličine samog objekta.)

Postoje i drugi metodi koje možete primeniti na objekat (a takođe i na bilo koju klasu ili referencu klase). Evo delimičnog spiska:

`ClassName` Kao rezultat daje string sa nazivom klase.

`ClassNameIs` Proverava naziv klase.

`ClassParent` Kao rezultat daje referencu na roditeljsku klasu.

`ClassInfo` Kao rezultat daje pokazivač na interni tip informacije u vreme izvršavanja klase (Run Time Type Information — RTTI), koji se razmatra u knjizi "Delphi priručnik za programere" (Delphi Developer's Handbook).

`ClassType` Kao rezultat daje referencu na klasu objekta (ovo se ne može primeniti direktno na klasu već samo na objekat).

`InheritsFrom` Testira da li je klasa izvedena (direktno ili indirektno) iz date osnovne klase (slično operatoru `is`).

`InstanceSize` Kao rezultat daje veličinu podataka o objektu.

Ovi metodi klase `TObject` su dostupni objektima svake klase, jer je klasa `TObject` roditeljska klasa za svaku klasu. Evo kako možemo upotrebiti ove metode da bismo pristupili informacijama klase:

```
procedure TSenderForm.ShowSender (Sender: TObject);
begin
  Memo1.Lines.Add ('Class Name: ' +
    Sender.ClassName);

  if Sender.ClassParent <> nil then
    Memo1.Lines.Add ('Parent Class: ' +
      Sender.ClassParent.ClassName);

  Memo1.Lines.Add ('Instance Size: ' +
    IntToStr (Sender.InstancesSize));
```

Kod proverava da li je `ClassParent` nil u slučaju da koristite instancu tipa `TObject`, koja nema osnovni tip. Možete upotrebiti druge metode da biste izvršili testiranje. Na primer, sledećim kodom možete proveriti da li je `Sender` objekat određenog tipa:

```
if Sender.ClassType = TButton then . . .
```

Takođe, sledećim testom možete proveriti da li parametar `Sender` odgovara datom objektu:

```
if Sender = Button1 then . . .
```

Svi ovi delovi koda su deo primera IfSender.

Umesto proveravanja određenog objekta ili klase, češće će biti potrebno proveriti tip kompatibilnosti objekta date klase, to jest, biće potrebno da proverite da li je klasa objekta data klasa ili je neka od njehih potklasa. Ovo će Vam omogućiti da znate da li možete operisati sa objektom metodima definisanim za klasu. Ovaj test se može izvršiti upotrebom metoda `InheritsFrom`, koji se takođe poziva kada koristite operator `is`. Sledeća dva testa su ekvivalentna:

```
if Sender.InheritsFrom (TButton) then . . .
if Sender is TButton then . . .
```

Sve ove tehnike su prikazane u primeru IfSender, koji sadrži samo jednu obradu događaja, nazvanu `ShowSender`, povezanu sa događajem `OnClick` od nekoliko kontrola: tri kontrole, poljem za potvrdu i poljem za izmene. Jedna od kontrola je zapravo Bitmap kontrola, objekat potklase `TButton`. Možete videti izlaz ovog programa u vreme izvršavanja na slici 4.1.

SLIKA 4.1 *Izlaz primera IfSender*

Prikazivanje informacija klase

Primer IfSender se može proširiti tako da prikazuje kompletnu listu osnovnih klasa. Kada načinite referencu klase, Vi možete dodati sve osnovne klase reference listi `ListParent` sledećim kodom:

```
with ListParent.Items do
begin
  Clear;
  while MyClass.ClassParent <> nil do
  begin
    MyClass := MyClass.ClassParent;
    Add (MyClass.ClassName);
  end;
end;
```

Primetićete da koristimo referencu klase na početku petlje `while`, koja testira nepostojanje osnovne klase (jer je roditeljska klasa `TObject`). Alternativno smo mogli da napišemo iskaz `while` na neki od narednih načina:

```
while not MyClass.ClassNameIs ('TObject')do...
while MyClass <> TObject do ...
```

Kod sa iskazom `with` koji se odnosi na listu `ListParent` je deo primera ClassInfo, koji prikazuje listu roditeljskih klasa i neke druge informacije o nekoliko komponenata VCL-a sa strane Standard Components Palette. Te komponente su ručno dodate dinamičkom nizu koji čuva klase i koji je deklarisan na sledeći način:

```
private
  ClassArray: array of TClass;
```

Kada program počne izvršavanje, niz se koristi za čuvanje svih naziva klasa u listi. Odabirom elementa liste izaziva se inicijalizacija njegove osnovne klase, kao što možete videti u izlazu programa na slici 4.2.

SLIKA 4.2 *Izlaz primera ClassInfo*

NAPOMENA

Dodatno proširenje primera može biti prikazivanje svih osnovnih klasa različitih komponenata hijerarhije. Da bih to učinio, ja sam kreirao VclHierarchy Wizard, koji možete pronaći na mom web sajtu. ■

VCL hijerarhija

VCL definiše brojne potklase klase `TObject`. Mnoge od tih klasa su zapravo potklase drugih potklasa, čime se formira veoma složena hijerarhija. Izuzev ukoliko niste zainteresovani za kreiranje novih komponenata, Vi ćete obično koristiti krajnje (terminal) klase ove hijerarhije — listove hijerarhije drveta. Ovo, uistinu, nije precizan opis, jer se neki listovi mogu dalje proširiti izvođenjem novih komponenata, a neke klase višeg nivoa se mogu direktno izvesti.

NAPOMENA

Delphi dokumentacija sadrži veliku sliku hijerarhije VCL klasa. Mada je njena veličina čini neupotrebljivom, ona može predstavljati dragocenu referencu za razumevanje hijerarhije VCL klasa. Ponoviću, možete, takođe, pronaći hijerarhiju VCL klasa na mom web sajtu. ■

VCL hijerarhiju možemo podeliti u tri glavne oblasti. To su komponente, generički objekti i izuzeci. U Delphi IDE-u se komponente mogu vizuelno menjati, tipično upotrebom Form Designera, dok se ostalim tipovima klasa pristupa iz izvornog koda. Pošto je za detaljan opis potrebno mnogo prostora, ovo poglavlje sadrži samo opšte informacije, uglavnom o komponentama, ali i o nekim važnim klasama VCL-a.

Komponente

Komponente su centralni deo Delphi aplikacija. Kada pišete program, Vi zapravo birate brojne komponente i definišete njihove interakcije. To je sve za Delphi vizuelno programiranje.

Postoje različite vrste komponenata u Delphiju. Većina komponenata je prikazana na Components Paletti, ali neke od njih (uključući `TForm` i `TApplication`) nisu. Komponente su potklase klase `TComponent`. Kao takve, mogu se usmeriti u `DFM` fajl (kako su izvedene iz klase `TPersistent`, koja sadrži informacije koje su potrebne za usmeravanje) i mogu sadržati

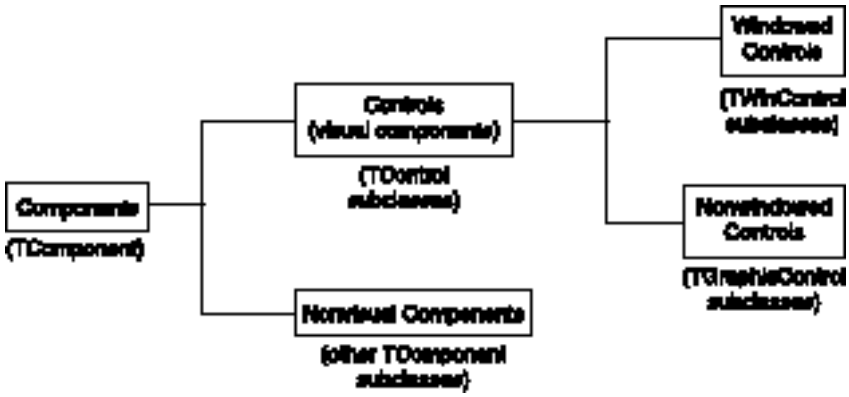
published svojstva i događaje kojima možete vizuelno manipulisati. Videli smo jednostavan primer (DateComp) izrade komponente u prethodnom poglavlju.

Deo VCL hijerarhije koji se odnosi na komponente je podeljen u tri oblasti, kao što možete videti na slici 4.3. Ove grupe predstavljaju komponente sa sličnom internom strukturom.

- **KONTROLE (CONTROLS) ILI VIZUELNE KOMPONENTE** (visual components) su klase koje su izvedene iz klase `TControl`. Kontrole imaju poziciju i veličinu na ekranu i prikazuju se na formularu u vreme dizajniranja na istom mestu koje će imati u vreme izvršavanja. Kontrole sadrže dve različite specifikacije, prema prozoru ili grafičke:
  - **KONTROLE PROZORA** (Window-based controls se, takođe, nazivaju kontrolama u okviru prozora — *windowed controls*) su vizuelne komponente zasnovane na prozoru operativnog sistema. S tehničke strane to znači da ove kontrole sadrže vezu sa prozorom i da su izvedene iz klase `TWinControl`. Sa gledišta korisnika kontrole u okviru prozora mogu dobiti ulazni fokus, a neke od njih mogu sadržati druge kontrole. One predstavljaju najveću grupu komponenata u Delphi VCL-u. Kontrole u okviru prozora možemo dalje podeliti u dve grupe: pakete Windows kontrola i uobičajene kontrole.
  - **GRAFIČKE KONTROLE** (graphical controls, takođe se nazivaju kontrolama koje nisu u okviru prozora — *nonwindowed controls*) su vizuelne komponente koje nisu zasnovane na prozoru. Zbog toga nemaju vezu sa prozorom, ne mogu dobiti fokus i ne mogu sadržati druge kontrole. Ove kontrole su izvedene iz klase `TGraphicControl` i iscratva ih roditeljski formular, koji im šalje događaje izazvane mišem i druge događaje. Primeri kontrola koje nisu u okviru prozora su komponente Label i SpeedButton. Postoji samo nekoliko kontrola u ovoj grupi, ali su kritične za minimizaciju upotrebe sistemskih resursa, naročito za komponente koje su brojne i često se koriste, kao što su oznake i kontrole palete alata.
- **NEVIZUELNE KOMPONENTE** (nonvisual components) su sve komponente koje nisu kontrole — sve klase izvedene iz klase `Tcomponent`, ali ne i iz klase `TControl`. U vreme dizajniranja obe komponente se na formularu prikazuju kao ikone (opciono se može prikazati i naslov komponente ispod ikone). U vreme izvršavanja, neke od ovih komponenata mogu da budu vidljive (na primer, standardni okviri za dijalog), dok su druge uvek nevidljive (na primer, komponenta tabele baze podataka). Drugim rečima, nevizuelne komponente nisu vidljive u vreme izvršavanja, mada mogu upravljati nečim što je vidljivo, recimo, okvirom za dijalog.

SAVET

Možete jednostavno pomeriti kursor iznad kontrole ili komponente u Form Designeru da biste prikazali pomoć sa nazivom i tipom klase. Možete, takođe, upotrebiti opciju okruženja, Show Component Captions, da biste videli naziv nevizuelne komponente odmah ispod njene ikone. ■



SLIKA 4.3 *Grafički prikaz grupa komponenata*

Windows komponente

Možda ste se pitali odakle je potekla ideja o upotrebi komponenata za Windows programiranje. Odgovor je jednostavan: sam Windows sadrži komponente koje se nazivaju kontrolma. Kontrola (control) je unapred određeni prozor koji ima specifično ponašanje i stilove, i ima mogućnost reagovanja na određene poruke. Ove kontrole su bile prvi korak u razvoju komponenata. Drugi korak su verovatno bile kontrole Visual Basic, a treći Delphi komponente.

NAPOMENA

Microsoftov treći korak je njegov ActiveX, koji predstavlja naslednika VBX kontrola. U Delphiju možete koristiti ActiveX i Delphi komponente, ali kada pogledate tehnologiju, Delphi komponente su zaista ispred ActiveX kontrola. Delphi komponente koriste OOP do krajnjih granica, dok ActiveX kontrole ne primenjuju u potpunosti koncept nasleđivanja. Detaljno ću obraditi upotrebu i stvaranje ActiveX kontrola u Poglavlju 16. ■

Windows 3.1 sadrži šest vrsta unapred određenih kontrola koje se uglavnom koriste u okvirima za dijalog. Za Win32 se i dalje koriste kontrole (buttons, push buttons, polja za potvrdu — check boxes i opcione kontrole — radio buttons), statičke oznake (static labels), polja za izmene (edit fields), liste (list boxes), kombo liste (combo boxes) i klizači (scroll bars). Win32 dodaje veliki broj novih unapred određenih komponenata, kao što su prikaz liste (list view), statusna linija (status line), roleri (spin buttons), linija napredovanja (progress bar), tabovi (tab control) i mnoge druge. Win32 programeri mogu koristiti standardne kontrole koje obezbeđuje sistem, dok Delphi programeri imaju prednost upotrebe odgovarajućih komponenata.

Standardne sistemske kontrole su osnovne komponente svake Windows aplikacije, bez obzira na programski jezik koji je upotrebljen da bi se napisale, i dobro su poznate svakom korisniku Windowsa. Delphi je bukvalno upakovao ove Windows komponente u neke od osnovnih komponenata. Delphi klasa, na primer klasa `TEdit`, jednostavno izvlači mogućnosti Windows kontrole koja se nalazi ispod nje, što olakšava upotrebu kontrole. Ipak, Delphi ništa ne dodaje mogućnostima takve kontrole. U Windowsu 95/98 kontrola za izmene ili tekst ima fizičko ograničenje od 32KB teksta, a to ograničenje ostaje i za Delphi komponentu.

Zbog čega Borland nije prevazišao ograničenje? Zbog čega ne možemo promeniti boju kontrole? Jednostavno zbog toga što bismo zamenom standardne kontrole korisničkom kontrolom izgubili blisku vezu sa operativnim sistemom. Pretpostavimo da Microsoft unapredi neke kontrole u narednoj verziji Windowsa. Da smo upotrebili našu verziju komponente, aplikacija koju smo izradili ne bi imala nove mogućnosti.

Upotrebom kontrola koje koriste mogućnosti operativnog sistema naši programi lako mogu prelaziti između različitih verzija operativnog sistema i zadržati sve svoje mogućnosti koje nudi svaka od verzija.

Naravno, ukoliko Vam je potrebna kontrola koja zaista radi nešto drugo od postojeće kontrole, potrebno je da napišete sopstvenu kontrolu, nešto što VCL čini u klasi izvedenoj iz klase `TCustomControl`. Na primer, Delphi tabela (grid) nije vezana za bilo koju Windows kontrolu. Sve klase u tom delu VCL drveta nisu direktno u vezi sa standardnim Windows kontrolama ili uobičajenim Win32 kontrolama.

Primetićete da je pakovanje postojećeg Windowsa efikasan način za ponovnu upotrebu koda, a takođe pomaže u smanjenju veličine Vašeg kompajliranog koda. Implementiranje potpuno nove kontrole zahteva Vaš kod u okviru Vaše aplikacije, dok upakovana kontrola operativnog sistema zahteva manje koda i čini upotrebljivim sistemski kod koji dele sve Windows aplikacije.

Objekti

Mada je VCL u osnovi kolekcija komponenata, postoje i druge klase koje ne pripadaju ovoj kategoriji jer nisu izvedene iz klase `TComponent`. Sve klase koje nemaju komponente često su (u Delphi Help fajlovima i dokumentaciji) označene kao *objekti*, mada to nije precizna definicija. Ove klase se mogu upotrebiti na dva osnovna načina. Uopšte, klase bez komponenata definišu tip podataka svojstava komponente, recimo svojstvo `Picture` komponente za slike (koja je `TGraphic` objekat) ili svojstvo `Items` liste (koja je `TStrings` objekat). Ove klase su izvedene iz klase `TPersistent`, te se mogu usmeravati (streamable) i mogu imati i podsvojstva, pa čak i događaje.

Drugi način upotrebe ovih klasa je direktna upotreba. U Delphi kodu koji pišete možete alocirati objekte tih klasa i manipulisati njima. To možete učiniti za brojne namene, uključujući čuvanje kopije vrednosti svojstva u memoriji i izmene a da ne promenite originalnu komponentu, za čuvanje liste vrednosti, za pisanje složenih algoritama i tako dalje. U ovoj knjizi ćete videti nekoliko primera koji prikazuju direktnu upotrebu klasa bez komponenata.

U VCL-u postoji nekoliko grupa ovakvih klasa.

- **GRAFIČKI OBJEKTI** (graphic-related objects) sadrže `TBitmap`, `TBrush`, `TCanvas`, `TFont`, `TGraphic`, `TGraphicObject`, `TIcon`, `TMetafile`, `TPen` i `TPicture`.
- **OBJEKTI TOKA/FAJLA** (stream/file-related objects) sadrže `TBlobStream`, `TFileStream`, `THandleStream`, `TIniFile`, `TMemoryStream`, `TFiler`, `TReader` i `TWriter`.
- **LISTE I KOLEKCIJE** (lists i collections) sadrže `TList`, `TStrings`, `TStringList`, `TCollection`, `TCollectionItem` i nove kontejner klase predstavljene u Delphiju 5. Pozabavićemo se ovim klasama u narednim odeljcima ovog poglavlja.

- **COM KLASE** (COM-related): Ovo je važna oblast Delphi programiranja. COM klasama ćemo se baviti u Poglavlju 15.
- **KLASE IZUZETAKA** (exception classes): Ove klase su izvedene iz klase `Exception`. Obradu izuzetaka smo razmatrali u Poglavlju 3, te ovde neću ponavljati detalje.

Zajednička VCL svojstva

Mada svaka kontrola sadrži svoj skup svojstava, možda ste primetili da su neka svojstva zajednička za sve kontrole. U tabeli 4.1 su prikazana neka od zajedničkih svojstava i njihovi kratki opisi.

Tabela 4.1: Neka svojstva koja postoje za većinu komponenata

Svojstvo	Postoji za	Opis
`Action`	Neke kontrole	Označava objekat Action povezan sa kontrolom (videti Poglavlje 5 za više detalja).
`Align`	Neke kontrole	Određuje kako je kontrola poravnata u roditeljskoj oblasti kontrole.
`Anchors`	Većinu kontrola	Označava stranu formulara sa kojom je vezana komponenta (videti Poglavlje 7 u kojem se nalazi primer).
`AutoSize`	Neke kontrole	Označava da li kontrola može sama odrediti svoju veličinu na osnovu sadržaja.
`BiDiMode`	Sve kontrole	Obezbeđuje podršku za jezike koji se pišu sleva nadesno (označava BiDirectional Mode — bidirekcioni mod).
`BorderWidth`	Kontrole u okviru prozora	Debljina bordure.
`BoundsRect`	Sve kontrole	Definiše okvirni četvorougao kontrole (samo u vreme izvršavanja).
`Caption`	Većinu kontrola	Naslov kontrole.
`Component Count`	Sve komponente	Broj komponenata koje poseduje aktuelna komponenta (samo u vreme izvršavanja i samo za čitanje).
`Component Index`	Sve komponente	Pozicija komponente u listi vlasnika komponente (samo u vreme izvršavanja).
`Components`	Sve komponente	Niz komponenata koje poseduje aktuelna komponenta (samo u vreme izvršavanja i samo za čitanje).
`Constraints`	Sve kontrole	Određuje minimalnu i maksimalnu veličinu kontrole (ili formulara) prilikom operacija promene veličine.
`ControlCount`	Sve kontrole	Broj kontrola koje poseduje aktuelna kontrola (samo u vreme izvršavanja i samo za čitanje).
`Controls`	Sve kontrole	Niz kontrola koje poseduje aktuelna kontrola (samo u vreme izvršavanja i samo za čitanje).
`Color`	Većinu kontrola	Određuje boju površine ili pozadine.

Tabela 4.1: Neka svojstva koja postoje za većinu komponenata

Svojstvo	Postoji za	Opis
`Ctrl3D`	Većinu komponenata	Određuje da li kontrola ima trodimenzionalni izgled.
`Cursor`	Sve kontrole	Kursor koji se koristi kada se pokazivač miša nalazi iznad kontrole.
`DockSite`	Većinu kontrola u okviru prozora	Označava da li kontrola u okviru prozora predstavlja mesto za dokiranje. Postoje i druga svojstva koja su u vezi sa ovim svojstvom, uključujući svojstva `DockClientCount`, `DockClients`, `UseDockManager` i `DockManager`. Dokiranje se razmatra u poglavljima 7 i 8.
`DragCursor`	Većinu kontrola	Kursor koji se koristi da bi se naznačilo da se kontrola može prevlačiti.
`DragKind`	Većinu kontrola	Omogućva Vam da odaberete prevlačenje ili dokiranje, ukoliko je mod prevlačenja automatski.
`DragMode`	Većinu kontrola	Određuje da li će se automatski aktivirati mod prevlačenja (omogućavajući prevlačenje ili dokiranje, što je naznačeno svojstvom `DragKind`).
`Enabled`	Sve kontrole i neke nevizuelne komponente	Određuje da li je kontrola aktivna ili ne (prikazana sivom bojom).
`Font`	Sve kontrole	Određuje font teksta koji se prikazuje unutar kontrole.
`Handle`	Sve kontrole u okviru prozora	Veza sa sistemskim prozorom koju koristi kontrola (samo u vreme izvršavanja i samo za čitanje).
`Height`	Sve kontrole	Vertikalna veličina kontrole.
`HelpContext`	Sve kontrole i komponente dijaloga	Kontekst broj koji se koristi za automatsko pozivanje ogovarajuće pomoći.
`Hint`	Sve kontrole	String koji se koristi za prikazivanje u oblačiću kontrole.
`Left`	Sve kontrole	Horizontalna koordinata gornjeg levog ugla komponente.
`Name`	Sve komponente	Jedinstveni naziv instance komponente, koji se može koristiti u izvornom kodu.
`Owner`	Sve komponente	Označava vlasnika komponente (samo u vreme izvršavanja i samo za čitanje).
`Parent`	Sve kontrole	Označava roditeljsku kontrolu (samo u vreme izvršavanja).
`ParentColor`	Većinu kontrola	Proverava da li komponenta koristi istu boju kao i roditeljska komponenta.
`ParentCtrl3D`	Većinu kontrola	Proverava da li komponenta koristi isti `Ctrl3D` kao i roditeljska komponenta.
`ParentFont`	Sve kontrole	Proverava da li komponenta koristi isti font kao i roditeljska komponenta.
`ParentShowHint`	Sve kontrole	Proverava da li komponenta koristi isti oblačić kao i roditeljska komponenta.

Svojstvo	Postoji za	Opis
PopupMenu	Sve kontrole	Iskačući meni koji se koristi kada korisnik klikne kontrolu desnim tasterom miša.
ShowHint	Sve kontrole	Proverava da li su oblačići aktivni.
Showing	Sve kontrole	Proverava da li se kontrola trenutno prikazuje na ekranu, to jest, ukoliko je aktivirano svojstvo Visible sve kontrole u roditeljskom lancu. Drugim rečima, kontrola se prikazuje (Showing) ukoliko je vidljiva, njena roditeljska kontrola je vidljiva, i svaka kontrola roditeljske kontrole je vidljiva, i tako dalje. (Samo u vreme izvršavanja i samo za čitanje).
za		
TabOrder	Sve kontrole u okviru prozora	Određuje redosled kontrole u okviru njene roditeljske kontrole.
TabStop	Sve kontrole u okviru prozora	Određuje da li korisnik može da pređe na kontrolu upotrebom tastera Tab.
Tag	Sve komponente	Celobrojna vrednost koja može da čuva nedefinisane podatke.
Top	Sve kontrole	Vertikalna koordinata gornjeg levog ugla komponente.
UndockHeight	Većinu kontrola	Visina kontrole kada nije dokirana.
UndockWidth	Većinu kontrola	Širina kontrole kada nije dokirana.
Visible	Sve kontrole	Određuje da li je kontrola vidljiva (ukoliko je i roditelj vidljiv, kao što je opisano za svojstvo Showing).
Width	Sve kontrole	Horizontalna veličina kontrole.

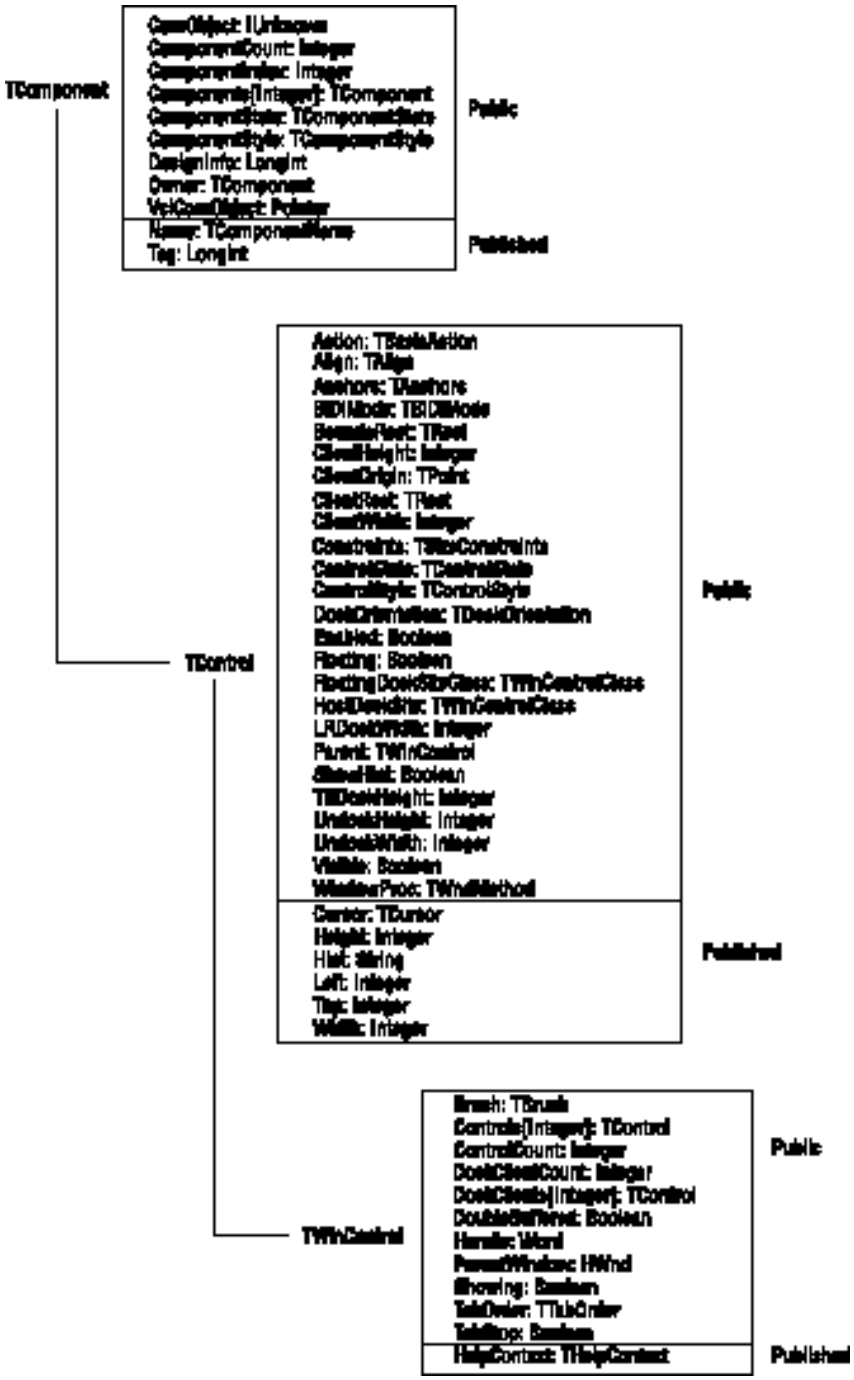
Pošto postoji nasleđivanje između komponenata, interesantno je videti u kojoj klasi pretku je predstavljeno najviše svojstava. Možete pogledati sliku 4.4 da biste videli pregled svojstava predstavljenih najvišim klasama VCL hijerarhije. Naredni odeljci daju opise ovih zajedničkih svojstava.

Svojstvo Name

Svaka komponenta u Delphiju bi trebalo da ima naziv. Naziv bi trebalo da bude jedinstven u okviru vlasnika komponente, što je uobičajeno formular na koji postavljate komponentu. To znači da aplikacija može da sadrži dva različita formulara, a da na svakome od njih bude komponenta sa istim nazivom, mada možda želite da izbegnete ovu praksu da biste izbegli konfuziju. Obično je bolje da nazivi komponenata budu jedinstveni u okviru aplikacije.

Određivanje odgovarajuće vrednosti za svojstvo Name je veoma važno. Ukoliko je naziv predugačak, moraćete da unosite dosta koda da biste koristili objekat; ukoliko je prekratak, možete pomešati različite objekte. Obično naziv komponente sadrži prefiks tipa komponente; ovim je kod mnogo čitljiviji i omogućava Delphiju da grupiše komponente u kombo kontroli Object Inspectora, gde su sortirane po nazivu. Postoje tri važna elementa vezana za svojstvo komponenata Name:

- Prvo, vrednost svojstva Name se koristi za definisanje naziva objekta u deklaraciji formulara klase. To je naziv koji ćete koristiti u kodu da biste se referisali na objekat. Zbog toga vrednost svojstva Name mora da bude ispravan Pascal identifikator.



SLIKA 4.4 *Svojstva koja su predstavljena najvišim klasama VCL hijerarhije i koja su dostupna u svim potklasama*

- Drugo, ukoliko odredite svojstvo `Name` pre nego što odredite svojstvo Caption, novi naziv se kopira za naslov. To jest, ukoliko su naziv i naslov jednaki, promena naziva će promeniti i naslov.
- Treće, Delphi koristi naziv komponente da bi kreirao unapred određeni naziv metoda događaja. Ukoliko imate komponentu `Button1`, njegova unapred određena obrada događaja `OnClick` biće nazvana `Button1Click`, izuzev ukoliko ne navedete drugačiji naziv. Ukoliko kasnije promenite naziv komponente, Delphi će promeniti nazive odgovarajućih metoda. Na primer, ukoliko promenite naziv kontrole u `MyButton`, metod `Button1Click` će automatski postati `MyButtonClick`.

Niz Components

Pored pristupanja komponenti preko njenog naziva, možete koristiti svojstvo `Components` njenog vlasnika, obično formulara. Evo primera koda koji možete upotrebiti za dodavanje liste naziva svih komponenata formulara (ovaj kod je deo primera ChangeOwner koji prikazujem u narednom odeljku):

```
procedure TForm1.Button1Click (Sender: TObject);
var
  I: Integer;
begin
  ListBox1.Items.Clear;
  for I := 0 to ComponentCount —1 do
    ListBox1.Items.Add (Componenta [I].Name;
end;
```

Ovaj kod koristi svojstvo `ComponentCount` u kojem se čuva ukupan broj komponenata koje poseduje formular, i svojstvo `Components` koje predstavlja spisak komponenata. Kada pristupite vrednosti liste, dobićete vrednost tipa `TComponent`. Zbog toga možete direktno koristiti samo svojstva zajednička za sve komponente, recimo svojstvo Name. Da biste koristili svojstva koja su specifična za komponentu, morate upotrebiti odgovarajuću kategorizaciju (`as`).

U Delphiju postoje komponente koje su ujedno i kontejneri komponenata: komponente GroupBox, Panel, PageControl i naravno Form. Kada koristite ove kontrole, možete u okviru njih dodati i druge kontrole. U tom slučaju kontejner je roditelj komponenata (što je naznačeno svojstvom `Parent`) dok je formular njihov vlasnik (što je naznačeno svojstvom `Owner`). Možete upotrebiti svojstvo `Controls` formulara ili GroupBoxa da biste se kretali kroz kontrole, a svojstvo `Components` formulara za kretanje kroz sve komponente, bez obzira na njihovog roditelja.

Koristeći svojstvo `Components` mi uvek možemo pristupiti svakoj komponenti formulara. Ukoliko Vam je potreban pristup određenoj komponenti, umesto da poredite svaki naziv komponente koju tražite, možete prepustiti Delphiju da to učini upotrebom metoda formulara `FindComponent`. Ovaj metod jednostavno pretražuje niz `Components`.

Svojstvo Owner

Svaka komponenta obično ima vlasnika. Kada se komponenta kreira u vreme dizajniranja (ili iz rezultujućeg DFM fajla), vlasnik komponente će neizbežno biti formular. Kada kreirate komponentu u vreme izvršavanja, vlasnik se prosleđuje kao parametar konstruktoru `Create`.

Svojstvo Owner je samo za čitanje, dakle ne možete ga promeniti. Ipak, možete uticati na vrednost svojstva pozivanjem metoda `InsertComponent` i `RemoveComponent` samog vlasnika, prosleđujući aktuelnu komponentu kao parametar. Upotrebom ovih metoda možete promeniti vlasnika komponente. Ipak, ne možete ih primeniti direktno u rutini obrade događaja formulara kao što smo mi to ovde pokušali:

```
procedure TForm1.ButtonClick (Sender: TObject);
begin
  RemoveComponent (Button1);
  Form2.InsertComponent (Button1);
end;
```

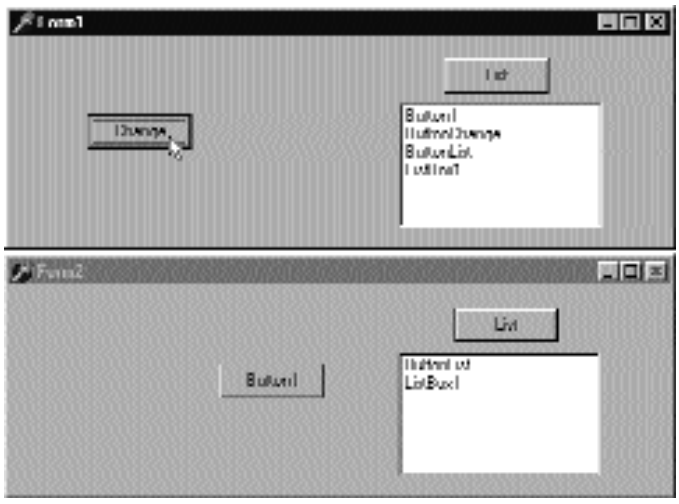
Ovaj kod dovodi do narušavanja pristupa memoriji, jer kada pozovete metod `RemoveComponent`, Delphi isključuje komponentu sa polja formulara (`Button1`) dodeljujući nil. Rešenje je da napišete proceduru kakva je ova:

```
procedure ChangeOwner (Component, NewOwner: TComponent);
begin
  Component.Owner.RemoveComponent (Component);
  NewOwner.InsertComponent (Component);
end;
```

Ovaj metod (izdvojen je iz primera ChangeOwner) menja vlasnika komponente. Poziva se uz jednostavniji kod da bi se promenila roditeljska komponenta promenom vlasnika:

```
procedure TForm1.ButtonChangeClick (Sender: TObject);
begin
  if Assigned (Buton1) then
  begin
    // change parent
    Button1.Parent := Form2;
    // change owner
    ChangeOwner (Button1, Form2);
  end;
end;
```

Ovaj metod proverava da li se polje `Button1` još uvek odnosi na kontrolu jer će prilikom pomeranja Delphi dodeliti `nil` za `Button1`. Na slici 4.5 možete videti efekat ovog koda.



SLIKA 4.5 *U primeru ChangeOwner, kada kliknete Change, komponenta Button1 će se premestiti na drugi formular*

Da bih pokazao da se `Owner` komponente `Button1` zaista menja, ja sam dodao još jednu funkciju za oba formulara. Kontrola List ispunjava polje nazivima komponenata koje poseduje svaki od formulara, a koristio sam proceduru prikazanu u prethodnom odeljku. Kliknite obe kontrole List pre i posle premeštanja komponente i videćete šta se dešava. Komponenta `Button1` ima jednostavnu rutinu za obradu događaja `OnClick` kojom se prikazuje naslov vlasnika formulara:

```
procedure TForm1.Button1Click (Sender: TObject);
begin
  ShowMessage ('My owner is ' +
    ((Sender as TButton).Owner as TForm).Caption);
end;
```

Uklanjanje polja formulara

Svaki put kada dodate komponentu formularu, Delphi dodaje kompletan opis, uključujući i sva svojstva, u DFM fajl. Pascal fajlu Delphi dodaje odgovarajuće polje u deklaraciju klase formulara. Kada je formular kreiran, Delphi učitava DFM fajl i koristi ga pri ponovnom kreiranju svih komponenata i pravilnom određivanju njihovih svojstava. Zatim povezuje novi objekat sa poljem formulara koje odgovara svojstvu Name.

Zbog toga je očigledno moguće postojanje komponente bez naziva. Ukoliko Vaša aplikacija ne manipuliše komponentom ili je ne menja u vreme izvršavanja, tada možete ukloniti naziv komponente iz Object Inspectora. Primeri su statičke oznake sa tekstom fiksne dužine ili element menija ili, što je još očiglednije, separatori menija. Uklanjanjem naziva Vi uklanjate odgovarajući element iz deklaracije klase formulara. Na ovaj način se smanjuje veličina objekta formulara (za samo četiri bajta što je veličina reference objekta) i smanjuje se DFM fajl jer se ne uključuje nepotrebni string (naziv komponente). Smanjenje DFM fajla dovodi do smanjenja veličine EXE fajla, iako je to samo malo.

UPOZORENJE

Ukoliko uklanjate nazive komponenata, postarajte se da ostavite bar jednu komponentu za svaku klasu na formularu tako da linker može da ih poveže u traženi kod. Ukoliko, kao primer, uklonite sa formulara sva polja koja se odnose na oznake, Delphi linker će ukloniti implementaciju klase `TLabel` iz izvršnog fajla. Kada sistem učita formular u vreme izvršavanja, neće moći da kreira objekat nepoznate klase i prikazaće poruku kojom Vas obaveštava da je klasa nedostupna. ■

Možete, takođe, zadržati naziv komponente i ručno ukloniti odgovarajuće polje iz klase formulara. Iako komponenta nema odgovarajuće polje formulara, ono se ipak kreira, mada će njegova upotreba biti nešto otežana (upotrebom metoda formulara `FindComponent`, na primer).

Sakrivanje polja formulara

Mnogi OOP puritanci se žale da Delphi ne primenjuje enkapsulaciju dosledno jer su sve komponente formulara mapirane preko javnih polja i može im se pristupiti iz drugih formulara i jedinica. Ipak, Delphi to čini samo da bi pomogao početnicima da brzo nauče kako da koriste Delphi vizuelno okruženje za programiranje. Programer može koristiti drugačiji pristup i koristiti svojstva i metode da bi operisao nad formularima. Rizik je što neki drugi programer istog tima može slučajno da zaobiđe ovaj pristup direktno pristupajući komponentama ukoliko su ostavljene u javnoj sekciji. Rešenje, za koje mnogi programeri ne znaju, je da premestite komponente u privatni deo deklaracije klase.

Kao primer, ja sam uzeo veoma jednostavan formular sa poljem za izmene, kontrolom i listom. Kada polje za izmene sadrži tekst i korisnik klikne kontrolu, tekst se dodaje listi. Kada je polje za izmene prazno, kontrola nije aktivna. Ovo je jednostavan kod primera HideComp:

```
procedure TForm1.Button1Click (Sender: TObject);
begin
  ListBox1.Items.Add (Edit1.Text);
end;
```

Ja sam naveo ove metode samo da bih Vam pokazao da se u kodu formulara obično referišemo samo na moguće komponente definišući njihove interakcije. Zbog toga izgleda nemoguće rešiti se polja koja odgovaraju komponenti. Ipak, ono što možemo učiniti je da ih sakrijemo, premeštajući ih iz unapred određenog javnog odeljka u privatni odeljak deklaracije klase formulara:

```
TForm1 = class (TForm)
  procedure Button1Click (Sender: TObject);
  procedure Edit1Change (Sender: TObject);
  procedure FormCreate (Sender: TObject);
private
  Button1: TButton;
  Edit1: TEdit;
  ListBox1: TListBox;
end;
```

Ako sada pokrenete program, upašćete u nevolje. Formular će se korektno učitati, ali pošto privatna polja nisu inicijalizovana, gore navedeni događaji će koristiti `nil` referencu objekta. Delphi obično inicijalizuje javna polja formulara koristeći komponente koje se kreiraju iz DFM fajla. Šta ukoliko mi to učinimo narednim kodom?

```
procedure TForm1.FormCreate (Sender: TObject);
begin
  Button1 := FindComponent ('Button1') as TButton;
  Edit1 := FindComponent ('Edit1') as TEdit;
  ListBox1 := FindComponent ('ListBox1') as TListBox;
end;
```

Ovo će gotovo funkcionisati, ali će generisati sistemsku grešku sličnu onoj koju smo razmatrali u prethodnom odeljku. Ovoga puta će privatne deklaracije prouzrokovati da linker poveže implementacije ovih klasa, ali problem je što sistem mora da zna nazive klasa da bi locirao reference koje su potrebne za konstruisanje komponenata prilikom učitavanja DFM fajla.

Poslednji korak koji nam je potreban je kod registracije da Delphiju u vreme izvršavanja naznači postojanje klasa komponenata koje želimo da koristimo. To bi trebalo da učinimo pre nego što se kreira formular, te ja taj kod obično smeštam u inicijalizacioni deo jedinice:

```
initialization
RegisterClasses ([TButton, TEdit, TListBox]);
```

Pitanje je sada da li je sve ovo vredelo truda? Ono što smo dobili je viši nivo enkapsulacije, štiteći komponente formulara od drugih formulara (i programera koji ih pišu). Moram da istaknem da ponavljanje ovih koraka za svaki formular može da bude dosadno, i zaista bih želeo da imam čarobnjaka koji bi ovaj kod generisao za mene dok ja obavljam standardne operacije u Delphiju. Ipak, za veliki projekat koji se izrađuje prema principima objektno orijentisanog programiranja, ja Vam preporučujem da razmislite o ovoj ili nekoj sličnoj tehnici.

Svojstva koja se odnose na veličinu i poziciju kontrole

Ostala važna svojstva, zajednička za sve kontrole, su ona svojstva koja se odnose na veličinu i poziciju. Pozicija kontrole je određena svojstvima `Left` i `Top`; veličina kontrole je određena svojstvima `Height` i `Width`. Sve komponente imaju poziciju jer kada ponovo otvorite postojeći formular u vreme izvršavanja, Vi želite da imate mogućnost da vidite ikone za nevizuelne komponente na poziciji na koju ste ih postavili. Ta pozicija je vidljiva iz DFM fajla.

Važna funkcija pozicije je da se, kao i svaka koordinata pod Windowsom, odnosi na oblast klijenta svoje roditeljske komponente (a to je komponenta koja je naznačena svojstvom `Parent`). Za formular, oblast klijenta je površina koja se nalazi unutar granica (isključujući same granice). Bilo bi nezgodno raditi sa ekranskim koordinatama, mada postoje neki već pripremljeni metodi koji konvertuju koordinate formulara i ekrana i obrnuto.

Imajte, ipak, na umu da su koordinate kontrole uvek relativne u odnosu na roditeljsku kontrolu, koja je obično formular, ali to može biti i panel ili neka druga *kontejnerska* komponenta. Ukoliko na formular postavite panel i na panel kontrolu, koordinate kontrole su relativne u odnosu na panel, a ne u odnosu na formular koji sadrži panel. Zapravo, u ovom slučaju roditeljska komponenta kontrole je panel.

Svojstva aktiviranja i vidljivosti

Postoje dva osnovna svojstva koja možete upotrebiti da biste omogućili korisniku da aktivira ili sakrije komponentu. Najjednostavnije od svih svojstava je svojstvo `Enabled`. Kada komponenta nije aktivna (kada je vrednost svojstva `Enabled False`), obično postoji neki vidljivi nagoveštaj kojim se ovo stanje prikazuje korisniku. U vreme dizajniranja svojstvo aktiviranja nema uvek nekakav efekat, ali u vreme izvršavanja neaktivne komponente su obično predstavljene sivom bojom.

U radikalnijem pristupu možete potpuno sakriti komponentu, bilo upotrebom odgovarajućeg metoda `Hide` ili dodeljivanjem vrednosti `False` svojstvu `Visible`. Pripazite jer provera vrednosti svojstva `Visible` ne govori uvek da li je kontrola vidljiva. Zapravo, ukoliko je kontejner kontrole sakriven, čak i kada je sama kontrola vidljiva, Vi je ipak ne možete videti. Zbog toga postoji još jedno svojstvo, svojstvo `Showing`, koje se može koristiti samo u vreme izvršavanja i samo za čitanje. Možete pročitati vrednost svojstva `Showing` da biste saznali da li je kontrola vidljiva korisniku, to jest, ukoliko je vidljiva, njena roditeljska kontrola je vidljiva, roditeljska kontrola roditeljske kontrole je vidljiva, i tako dalje.

Prilagodljivo svojstvo Tag

Svojstvo `Tag` je čudno jer nema nikakav efekat. To je samo dodatna memorijska lokacija, prisutna u svakoj klasi komponenata, gde možete sačuvati zajedničke vrednosti. Vrsta informacije i način na koji ćete je upotrebiti je prepušten Vama.

Često je korisno imati dodatnu memorijsku lokaciju da biste dodelili informaciju komponenti, a da ne morate da definišete sopstvenu klasu komponenata. Tehnički, svojstvo `Tag` čuva celobrojnu vrednost (long integer) tako da, na primer, možete čuvati bilo broj koji je deo niza, bilo listu koja odgovara objektu. Koristeći konverziju tipova u svojstvu Tag možete čuvati pokazivač, objekat ili bilo šta drugo što je dužine četiri bajta. Ovo omogućava programeru da uz komponentu asocira bilo šta koristeći tag komponente. Kako da upotrebimo ovo svojstvo, videćemo u primerima narednih poglavlja, uključujući primere ODMenu u Poglavlju 5.

Korisnički interfejs: boja i font

Dva svojstva koja se često koriste za prilagođavanje korisničkog interfejsa su svojstva `Color` i `Font`. Postoji nekoliko svojstava koja su vezana za boju. Svojstvo `Color` se obično odnosi na boju pozadine komponente. Takođe, postoji svojstvo Color za fontove i mnoge druge grafičke elemente. Mnoge komponente imaju, takođe, svojstva `ParentColor` i `ParentFont`, koja označavaju da li komponenta treba da koristi boju i font roditeljske komponente, koja je obično formular. Možete upotrebiti ova svojstva da biste promenili font svake od kontrola formulara određivanjem samo svojstva `Font` samog formulara.

Kada određujete font, bilo unošenjem vrednosti atributa svojstva u Object Inspectoru bilo upotrebom standardnog okvira za dijalog za izbor fonta, možete odabrati jedan od fontova instaliranih na sistemu. Činjenica da Vam Delphi omogućava sve fontove instalirane na sistemu ima prednosti i nedostatke. Glavna prednost je to što imate veliki broj lepih fontova, a Vaš program može upotrebiti bilo koji od njih. Nedostatak je to da ukoliko distribuirate Vašu aplikaciju, ti fontovi možda nisu dostupni na kompjuterima korisnika.

Ukoliko Vaš program koristi font koji korisnik ne poseduje, Windows će odabrati neki drugi font umesto upotrebljenog fonta. Pažljivo formatirani izlaz programa može biti uništen zamenom fontova. Zbog toga bi trebalo da se oslonite samo na standardne fontove Windowsa (kao što su MS Sans Serif, System, Arial, Times New Roman i tako dalje). Alternativa je da obezbedite fontove uz Vašu aplikaciju, ukoliko Vam to dopušta licenca za upotrebu fontova.

Postoji veliki broj načina za određivanje boje. Tip ovog svojstva je TColor. Za svojstva ovog tipa možete odabrati vrednost iz niza unapred određenih naziva ili možete direktno uneti vrednost. Konstante boja uključuju `clBlue`, `clSilver`, `clWhite`, `clGreen`, `clRed` i mnoge druge. Još bolji način koji možete upotrebiti je da koristite boje koje koristi Windows za svoje elemente, kao što je pozadina prozora (`clWindow`), boja istaknutog teksta menija (`clHighlightText`), boja aktivnog naslova (`clActiveCaption`) ili univerzalna boja kontrole (`clBtnFace`). Sve konstante boja koje su ovde pomenute su date u Delphi Helpu pod temom *TColor type*.

Druga mogućnost je da `TColor` naznačite brojem (četvorobajtna heksadecimalna vrednost) umesto da koristite unapred određenu vrednost. Ukoliko koristite ovaj pristup, trebalo bi da znate da najniža tri bajta ovog broja predstavljaju RGB intenzitet boja za plavu, zelenu i crvenu, respektivno. Na primer, vrednost $00FF0000 odgovara čistoj plavoj boji, vrednost $0000FF00 čistoj zelenoj boji, vrednost $000000FF čistoj crvenoj boji, vrednost $00000000 crnoj boji, a vrednost $00FFFFFF beloj boji. Određivanjem bilo koje međuvrednosti možete dobiti bilo koju od 16 miliona mogućih boja.

Umesto da direktno određujete ove heksadecimalne vrednosti, trebalo bi da upotrebite funkciju `RGB` koja ima tri parametra, sva tri u opsegu od 0 do 255. Prvi određuje količnu crvene boje, drugi određuje količinu zelene boje, a poslednji količinu plave boje. Upotreba funkcije `RGB` čini program čitljivijim nego kada koristite heksadecimalne konstante.

NAPOMENA

RGB je gotovo Windows API funkcija. Definisana je Windows jedinicama a ne Delphi jedinicama, ali slična funkcija ne postoji u Windows API-ju. U C-u postoji makro koji ima isti efekat, te je ovo dobrodošao dodatak Pascal interfejsu za Windows. ■

Najviši bajt tipa `TColor` se koristi za označavanje palete koju bi trebalo pretražiti u potrazi za bojom koja najbolje odgovara boji koja je tražena, ali palete su suviše napredna tema da bismo je ovde razmatrali. (Sofisticirani programi za obradu slika takođe koriste ovaj bajt da bi preneli informaciju transparentnosti za svaki element koji se prikazuje na ekranu.) U vezi sa odgovarajućim paletama i bojama, imajte na umu da Windows ponekad zamenjuje odabranu boju punom bojom koja je najbliža izgledom, bar u video modovima koji koriste paletu. To je uvek slučaj sa fontovima, linijama i drugim. U ostalim slučajevima Windows koristi tehniku umekšavanja, stvaranja međutonova (dithering) da bi oponašao traženu boju iscrtavajući gustu mrežu piksela od mogućih boja. Kod 16-bitnih (VGA) adaptera pri velikim rezolucijama često ćete videti čudne mreže piksela različitih boja, a ne boju koju ste imali na umu.

Zajednički VCL metodi

Metodi komponente su kao bilo koji drugi metodi. Postoje procedure i funkcije koje možete pozvati da biste izvršili odgovarajuću akciju. Kao što je ranije pomenuto, možete često upotrebiti metode da izvedete isti efekat kao kada biste pročitali ili zapisali svojstvo. Kada koristite svojstva, obično je lakše zapisati i razumeti kod. Ipak, nemaju svi metodi odgovarajuća svojstva. Većina od njih su procedure, koje izvršavaju akciju umesto čitanja ili zapisivanja vrednosti. Ponovimo, neki metodi su dostupni za sve komponente; ostale metode dele samo komponente (vizuelne komponente) i tako dalje. Tabela 4.2 prikazuje neke zajedničke metode komponenata. U knjizi ćemo videti primere koji koriste većinu ovih metoda.

Tabela 4.2: Neki metodi koji su dostupni za većinu VCL komponenata

Metod	Dostupan za	Opis
`BeginDrag`	Sve kontrole	Započinje ručno prevlačenje.
`BringToFront`	Sve kontrole	Pomera kontrolu ispred svih ostalih.
`CanFocus`	Sve kontrole	Određuje da li će kontrola prihvatiti ulazni fokus sa tastature.
`ClientToScreen`	Sve kontrole	Prevodi koordinate klijenta u koordinate ekrana.
`ContainsControl`	Sve kontrole	Određuje da li aktuelna kontrola sadrži određenu kontrolu.
`Create`	Sve komponente	Kreira novu instancu (konstruktor).
`Destroy`	Sve komponente	Uklanja instancu (destruktor). Trebalo bi, zapravo, da pozovete `Free`.
`Dragging`	Sve kontrole	Označava da li se kontrole prevlače.
`EndDrag`	Sve kontrole	Ručno prekida prevlačenje.
`ExecuteAction`	Sve komponente	Aktivira akciju povezanu sa komponentom.
`FindComponent`	Sve komponente	Daje kao rezultat komponentu iz niza `Components` koja ima dati naziv (koristili smo je samo u primeru HideComp).
`FlipChildren`	Sve kontrole u okviru prozora	Premešta dete kontrole sa leve na desnu stranu i obrnuto. Koristi se za podršku jezicima koji se pišu zdesna nalevo (recimo arapski i hebrejski), uz svojstvo `IsRightToLeft`.
`Focused`	Sve kontrole u okviru prozora	Određuje da li kontrola ima fokus.
`Free`	Sve komponente	Uklanja objekat iz memorije (za formular bi trebalo koristiti metod Release).
`GetTextBuf`	Sve kontrole	Daje tekst (ili naslov) kontrole.
`GetTextLen`	Sve kontrole	Kao rezultat daje dužinu teksta (ili naslova) kontrole.
`HandleAllocated`	Sve kontrole	Kao rezultat daje True ukoliko je alocirano sistemsko upravljanje za kontrolu.
`HandleNeeded`	Sve kontrole	Postavlja odgovarajuće sistemsko upravljanje ukoliko takvo ne postoji.

Metod	Dostupan za	Opis
`Hide`	Sve kontrole	Čini kontrolu vidljivom (jednako kao i dodeljivanje vrednosti `False` svojstvu `Visible`).
`InsertComponent`	Sve komponente	Dodaje novi element listi komponenata koje poseduje.
`InsertControl`	Sve kontrole	Dodaje novi element listi kontrola koje pripadaju aktuelnoj kontroli.
`Invalidate`	Sve kontrole	Zahteva ponovno iscrtavanje kontrole.
`ManualDock`	Sve kontrole	Ručno aktivira dokiranje.
`ManualFloat`	Sve kontrole	Dokirana kontrola postaje pokretna.
`Remove Component`	Sve komponente	Uklanja komponentu iz liste `Components`.
`ScaleBy`	Sve kontrole	Menja veličinu kontrole za dati procenat.
`ScreenToClient`	Sve kontrole	Prevodi ekranske koordinate u koordinate klijenta.
`ScrollBy`	Sve kontrole	Pomera sadržaj kontrole.
`SendToBack`	Sve kontrole	Pomera kontrolu iza svih ostalih kontrola.
`SetBounds`	Sve kontrole	Menja poziciju i veličinu kontrole (brže nego da svojstvima pristupate pojedinačno).
`SetFocus`	Sve kontrole	Kontroli dodeljuje ulazni fokus.
`SetTextBuf`	Sve kontrole	Određuje tekst (ili naslov) kontrole.
`Show`	Sve kontrole	Čini kontrolu vidljivom (isto kao kada biste dodelili vrednost True svojstvu `Visible`).
`Update`	Sve kontrole	Momentalno ponovo iscrtava kontrolu, ukoliko postoje zahtevi za iscrtavanje koji čekaju izvršenje.

Zajednički VCL događaji

Kao što postoje zajednička svojstva za sve komponente, postoje i neki događaji koje možete upotrebiti za svaku od njih. U tabeli 4.3 je dat kratak opis ovih događaja. Ponovimo, ova tabela bi trebalo da posluži samo kao polazna tačka. U knjizi ćete videti primere koji koriste većinu ovih događaja.

Tabela 4.3: Neki događaji koji su dostupni za većinu komponenata

Događaj	Dostupan za	Opis
OnCanResize	Mnoge kontrole	Dešava se kada kontrola menja veličinu i omogućava prekid ove operacije.
OnChange	Mnoge komponente	Dešava se kada se objekat ili podaci menjaju.
OnClick	Većinu kontrola	Dešava se kada korisnik klikne levim tasterom miša iznad kontrole.
OnContext PopupMenu	Sve kontrole (novo u Delphiju 5)	Dešava se kada korisnik klikne kontrolu desnim tasterom miša. Omogućava Vam da obavite i neku drugu akciju, a ne samo prikazivanje iskačućeg menija.
OnDbClick	Mnoge kontrole	Dešava se kada korisnik dva puta klikne mišem iznad kontrole.
OnDockDrop	Kontrole u okviru prozora	Dešava se kada se operacija dokiranja završi iznad aktuelne kontrole.
OnDockOver	Kontrole u okviru prozora	Dešava se kada korisnik prevuče pokazivač miša iznad komponente prilikom operacije dokiranja.
OnDragDrop	Većinu kontrola	Dešava se kada se operacija prevlačenja završi iznad kontrole; šalje ga komponenta koja prima operaciju prevlačenja.
OnDragOver	Većinu kontrola	Dešava se kada korisnik prevuče pokazivač miša iznad komponente.
OnEndDock	Većinu kontrola	Dešava se kada se završi operacija dokiranja aktuelne komponente.
OnEnter	Sve kontrole u okviru prozora	Dešava se kada se komponenta aktivira, to jest, kada komponenta dobije fokus.
OnExit	Sve kontrole u okviru prozora	Dešava se kada komponenta izgubi fokus.
OnGetSiteInfo	Kontrole u okviru prozora	Kao rezultat daje informaciju o dokiranju komponente.
OnKeyDown	Neke kontrole u okviru prozora	Dešava se kada korisnik pritisne taster tastature; šalje se komponenti koja ima ulazni fokus.
OnKeyPress	Neke kontrole u okviru prozora	Dešava se kada korisnik pritisne taster; šalje se komponenti koja ima ulazni fokus.
OnKeyUp	Neke kontrole u okviru prozora	Dešava se kada korisnik otpusti taster; šalje se komponenti koja ima ulazni fokus.
OnMouseDown	Većinu kontrola	Dešava se kada korisnik pritisne neki od tastera miša; šalje se komponenti koja se nalazi ispod pokazivača miša.
OnMouseMove	Većinu kontrola	Dešava se kada korisnik pomera miša iznad komponente; šalje se komponenti ispod pokazivača miša.
OnMouseUp	Većinu kontrola	Dešava se kada korisnik otpusti neki od tastera miša; šalje se komponenti ispod pokazivača miša.

Događaj	Dostupan za	Opis
`OnMouseWheel`, `OnMouseWheelDown`, `OnMouseWheelUp`	Kontrole u okviru prozora	Dešava se kada korisnik pokreće točkić miša ili kada klikne točkić kao da je taster.
`OnResize`	Većinu kontrola	Dešava se kada se završi operacija promene veličine.
`OnStartDock`	Većinu kontrola	Dešava se kada korisnik započne dokiranje.
`OnStartDrag`	Većinu kontrola	Dešava se kada korisnik započne prevlačenje; šalje se komponenti koja započinje operaciju prevlačenja.
`OnUnDock`	Kontrole u okviru prozora	Dešava se kada se neka kontrola odvoji od aktuelne kontrole.

Razumevanje okvira

Poglavlje 1 je predstavilo okvire (frames) kao jednu od novih mogućnosti u Delphiju 5. Videli smo da možete da kreirate novi okvir, postavite na njega komponente, napišete obradu događaja za komponente i da zatim dodate okvir formularu. Drugim rečima, okvir je sličan formularu, ali definiše samo deo prozora a ne ceo prozor. To zaista nije mogućnost koja je vredna nove konstrukcije. Potpuno novi element okvira je to što možete kreirati više instanci okvira u vreme dizajniranja i što možete izmeniti klasu i instancu istovremeno. Ovo čini okvire efikasnim alatom za kreiranje prilagodljivih složenih kontrola u vreme dizajniranja, nešto blisko vizuelnom alatu za izradu komponenata.

Verovatno ste upoznati sa Delphi konceptom vizuelnog nasleđivanja formulara (razmatrano u Poglavlju 2). Možete raditi i sa osnovnim formularom i sa izvedenim formularom u vreme dizajniranja, a izmene osnovnog formulara će biti primenjene i na izvedeni formular, izuzev ukoliko ovo ne zaobilazi neko svojstvo ili događaj. Kada su okviri u pitanju, Vi radite sa klasom (što je uobičajeno za Delphi), ali razlika je u tome što takođe možete prilagoditi jednu ili više instanci klase koja je kreirana u vreme dizajniranja. Kada radite sa formularom, ne možete promeniti svojstvo klase `TForm1` objekta `Form1` u vreme dizajniranja. Kada su u pitanju okviri, to je moguće.

Kada jednom shvatite da radite sa klasom i jednom ili više njenih instanci u vreme dizajniranja, ne postoji ništa više što bi trebalo da shvatite, a ima veze sa okvirima. U praksi, okviri su korisni kada želite da upotrebite grupu komponenata za više formulara u okviru aplikacije. U tom slučaju, zapravo, možete prilagoditi svaku od instanci u vreme dizajniranja. Nije li to već bilo moguće sa šablonima komponenata? Jeste, ali šabloni komponenata su bazirani na konceptu kopiranja komponenata i njihovog koda. Nije postojao način za promenu originalne definicije šablona i da pogledate efekat na svakom mestu gde se koristi. To je ono što se dešava sa okvirima (i na drugačiji način sa vizuelnim nasleđivanjem formulara); promene originalne verzije (klase) se odslikavaju na kopije (instance).

Postoje i mnge druge primene okvira, što će biti očiglednije kada programeri Delphija prihvate ovu mogućnost. Okviri mogu biti veoma korisni prilikom izrade više strana na formularu što ću prikazati u Poglavlju 8.

Obratimo pažnju na još nekoliko elemenata okvira u primeru nazvanom Frames2. Ovaj primer sadrži okvir sa listom, poljem za izmene i tri kontrole za koje je napisan jednostavan kod koji operiše sa komponentama. Okvir je takođe poravnat sa oblašću klijenta i udubljen jer okviri nemaju borduru. Ovo je definicija okvira u njegovom DFM fajlu:

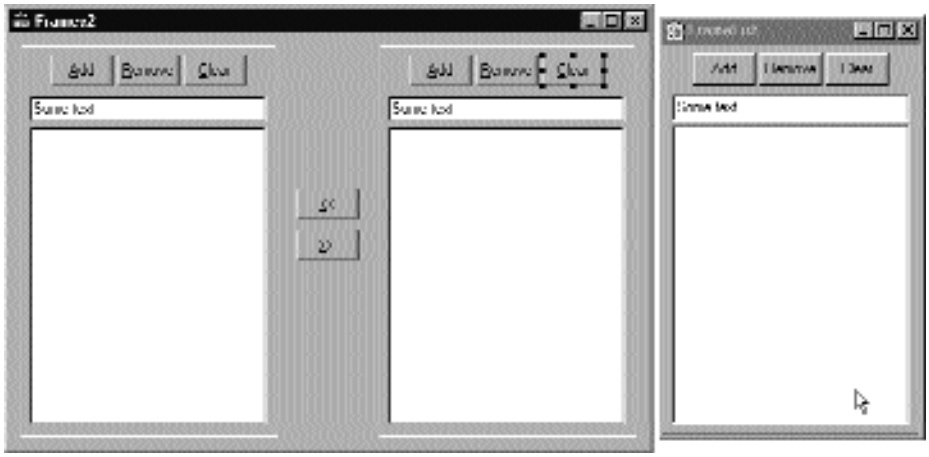
```
object FrameList: TFrameList
  Left = 0
  Top = 0
  Widtn = 202
  Height = 306
  Tab0rder = 0
  object Bevel: TBevel
    Align = alClient
    Shape = bsFrame
  end
  object ListBox: TListBox. . .
  object Edit: TEdit
    Text = 'Some text'
  end
  object btnAdd: TButton
    Caption = '&Add'
    OnClick = btnAddClick
  end
  object btnRemove: TButton
    Caption = '&Remove'
    OnClick = btnRemoveClick
  end
  object btnClear: TButton
    Caption = '&Clear'
    OnClick = btnClearClick
  end
end
```

Naravno, okvir ima odgovarajuću klasu, koja izgleda kao normalna klasa formulara:

```
type
  TFrameList = class(TFrame)
    ListBox: TListBox;
    Edit: TEdit;
    btnAdd: TButton;
    btnRemove: TButton;
    btnClear: TButton;
    Bevel: TBevel;
    procedure btnAddClick(Sender: TObject);
    procedure btnRemoveClick(Sender: TObject);
    procedure btnClearClick(Sender: TObject);
  private
    { Private declarations}
  public
    { Public declarations }
  end;
```

Razlika je u tome što okvir možete dodati formularu. Ja sam koristio dve instance okvira u primeru (kao što možete videti na slici 4.6) i malo sam izmenio ponašanje. Prva instanca okvira sadrži listu u kojoj su elementi sortirani. Kada promenite svojstvo komponente okvira, DFM fajl formulara će prikazati razlike, kao što to čini kada je u pitanju vizuelno nasleđeivanje formulara:

```
object FormFrames: TFormFrames
  Caption = 'Frames2'
  inline FrameListl: TFrameList
    Left = 8
    Top = 8
    inherited ListBox: TListBox
      Sorted = True
  end
end
inline FrameList2: TFrameList
  Left = 232
    Top = 8
    inherited btnClear: TButton
      OnClock = FrameList2btnClearClick
    end
  end
end
```



SLIKA 4.6 *Okvir i njegove dve instance u vreme dizajniranja u primeru Frames2*

Kao što iz listinga možete videti, DFM fajl za formular koji sadrži formulare koristi novu DFM ključnu reč, `inline`. Reference modifikovanih komponenata okvira umesto toga koriste ključnu reč `inhereted`, mada se ovaj termin koristi u proširenom značenju. Ovde se `inhereted` ne odnosi na osnovnu klasu iz koje se izvodi, već na klasu iz koje se kreira instanca objekta. Verovatno je bila dobra ideja da se upotrebi postojeća mogućnost vizuelnog nasleđivanja formulara i da se primeni u novom kontekstu. Efekat ovog pristupa, zapravo, je to što možete upotrebiti komandu Revert to Inhereted Object Inspectora ili formulara da biste poništili izmene i vratili unapred određene vrednosti svojstava.

Primetićete, takođe, da se nepromenjene komponente klase okvira ne nalaze u DFM fajlu formulara koji koristi okvir, i da formular sadrži dva različita okvira, ali da komponente oba okvira imaju jednake nazive. Zapravo, formular nije vlasnik ovih komponenata već su okviri vlasnici ovih komponenata. To znači da formular — da bi se obratio ovim komponentama — mora da im se obrati preko okvira, kao što možete videti u kodu za kontrole koje kopiraju elemente iz jedne liste u drugu.

```
procedure TFormFrames.btnLeftClick (Sender: TObject);
begin
  FrameList1.ListBox.Items.AddStrings (
    FrameList2.ListBox.Items);
end;
```

Konačno, pored izmena svojstava bilo koje instance okvira, možete promeniti i kod bilo koje obrade događaja. Ukoliko samo dva puta kliknete jednu od kontrola okvira dok radite nad formularom (ne sa samostalnim okvirom), Delphi će generisati ovakav kod za Vas:

```
procedure TFormFrames.FrameList2btnClearClick (Sender: TObject);
begin
  FrameList2.btnClearClick (sender);

end;
```

Linija koda koju Delphi automatski dodaje za Vas odgovara pozivu nasleđene obrade događaja osnovne klase vizuelnog nasleđivanja formulara. Ovoga puta, ipak, da biste dobili unapred određeno ponašanje okvira, potrebno je pozvati obradu događaja i primeniti je na određenu instancu — na sam objekat okvira. Aktuelni formular, zapravo, ne uključuje ovu obradu događaja i o njoj ništa ne zna.

Da li ćete postaviti ovaj poziv ili ćete ga ukloniti, zavisi od onoga što želite da učinite. Ja sam u primeru odlučio da uslovno izvršim unapred određeni kod, u zavisnosti od konfirmacije korisnika:

```
procedure TFormFrames.FrameList2btnClearClick (Sender: TObject);
begin
  if (MessageDlg ('OK to empty the list box?',
    mtConfirmation, [mbYes, mbNo], 0) = idYes then
  // execute standard frame code
  FrameList2.btnClearClick (Sender);
end;
```

SAVET

Usput primetite da pošto obrada događaja sadrži nekakav kod, ostavljanje prazne obrade događaja i čuvanje formulara neće ukloniti kod, što je uobičajeno; zapravo, obrada događaja nije prazna! Ukoliko jednostavno želite da izostavite unapred određeni kod događaja, potrebno je da dodate bar jedan komentar da biste izbegli da sistem automatski ukloni obradu događaja! ■

Klase lista i kontejnera

Često je važno obraditi grupe komponenata ili objekata. Pored upotrebe standardnih nizova i dinamičkih nizova, postoji nekoliko VCL klasa koje predstavljaju liste drugih objekata. Ove klase možemo podeliti u tri grupe: jednostavne liste, kolekcije i kontejnere. Poslednja grupa je predstavljena u Delphiju 5. Liste su predstavljene generičkom listom objekata, `TList`, i dvema listama stringova, `TStrings` i `TstringList`.

- `TList` definiše listu pokazivača koji se mogu upotrebiti za čuvanje objekata bilo koje klase. `TList` je fleksibilniji od dinamičkog niza jer se automatski proširuje, jednostavno dodavanjem novih elemenata. Prednost dinamičkih nizova nad `Tlistom` je u tome što Vam dinamički nizovi omogućavaju da odredite specifični tip objekata i obavite odgovarajuću proveru tipova prilikom kompajliranja.
- `TStrings` je apstraktna klasa koja predstavlja sve oblike lista stringova bez obzira na implementaciju načina čuvanja. Ova klasa definiše apstraktnu listu stringova. Zbog toga se objekti `TStrings` koriste samo kao svojstva komponenata koje mogu da čuvaju stringove, kao što su, recimo, liste.
- `TStringList`, potklasa `TStrings`, definiše listu stringova sa sopstvenim načinom čuvanja. Možete koristiti ovu klasu za definisanje liste stringova u programu.

Druga grupa, kolekcije, sadrži samo dve klase, `TCollection` i `TCollectionItem`. `TCollection` definiše homogenu listu objekata za koju je vlasnik klasa kolekcije. Objekti kolekcije moraju biti naslednici klase `TCollectionItem`. Ukoliko Vam je potrebna kolekcija koja čuva specifične objekte, morate kreirati potklasu `TCollection` i potklasu `TCollectionItem`. Kolekcije se uvek koriste za određivanje vrednosti svojstava komponenata. Nije uobičajeno raditi sa kolekcijama direktno u programu. Sve ove liste imaju veliki broj metoda i svojstava. Sa listama možete operisati upotrebom notacije za nizove (`"["` i `"]"`) kako za čitanje tako i za izmene elemenata. Postoji svojstvo Count, kao i tipični metodi pristupa, kao što su `Add, Insert, Delete, Remove` i metodi pretraživanja (na primer `IndexOf`).

Objekti `TStringList` i `TStrings` sadrže i listu stringova i listu objekata asociranih uz stringove. Time se otvara veliki broj različitih upotreba ovih klasa. Na primer, možete ih koristiti za rečnike asociranih obejkata ili za čuvanje bitmapa ili drugih elemenata koji se koriste u listama.

NAPOMENA

Komponenta `TListbox` zapravo koristi objekat `TStringList` kada je potrebno da sačuva stringove, kada je njen handle prozora nepravilan; koristi drugačijeg naslednika objekta `TStrings` kada konačno izvrši asocijaciju sa Windows list kontrolom koja čuva svoje stringove. ■

Dve klase liste stringova već imaju spremne metode za čuvanje ili učitavanje njihovog sadržaja u ili iz tekstualnog fajla, `SaveToFile` i `LoadFromFile`. Cikličnim prolaskom kroz listu možete upotrebiti jednostavan `for` iskaz nad indeksom, kao da je lista niz.

Upotreba liste objekata

Možemo napisati primer koji ističe upotrebu generičke klase `TList`. Kada Vam je potrebna lista bilo kakve vrste podataka, Vi možete deklarisati objekat `TList`, popuniti ga podacima, a zatim pristupiti podacima dok vršite pravilnu konverziju tipova. Primer ListDemo demonstrira baš to. Takođe prikazuje zamke ovakvog pristupa. Formular primera sadrži privatnu promenljivu koja čuva listu datuma:

```
private
  ListDate: TList;
```

Ovaj objekat liste se kreira kada se kreira i sam formular:

```
procedure TForm1.ButtonAddClick (Sender: TObject);
begin
  Radnomize;
  ListDate := TList.Create;
end;
```

Kontrola formulara dodaje slučajan datum listi (naravno, ja sam u projekat uključio i jedinicu koja sadrži komponentu datuma izrađenu u prethodnom poglavlju):

```
procedure TForm1.ButtonAddClick (Sender: TObject);
begin
  ListDate.Add (TDate.Create (1900 + Random (200),
    1 + Random (12), 1 + Random (30)));
end;
```

Kada izdvojite elemente iz liste, morate ih konvertovati u odgovarajući tip, kao u narednom metodu, koji je povezan sa kontrolom List (efekat možete videti na slici 4.7):

```
procedure TForm1.ButtonListDateClick (Sender: TObject);
var
  I: Integer;
begin
  ListBox1.Clear;
  for I := 0 to ListDate.Count —1 do
    ListBox1.Items.Add ((
      TObject(ListDate [I]) as TDate).Text);
end;
```

SLIKA 4.7 *Lista datuma prikazana primerom ListDemo*

Na kraju prethodnog koda, pre nego što možemo upotrebiti as, potrebno je da prvo opštije konvertujemo pokazivač koji TList šalje u referencu TObject. Ova vrsta izraza može dovesti do pogrešnog izuzetka konverzije, ili može generisati memorijsku grešku kada pokazivač nije referenca na objekat.

Da bih pokazao da stvari zaista mogu poći naopako, ja sam dodao još jednu kontrolu koja dodaje objekat TButton listi:

```
procedure TForm1.ButtonWrongClick (Sender: TObject);
begin
  // add a button to the list
  ListDate.Add (Sender);
end;
```

Ukoliko kliknete ovu kontrolu, a zatim ažurirate jednu od lista, dobićete grešku. Konačno, ne zaboravite da kada uklonite listu objekata, ne smete zaboraviti da prvo uklonite sve objekte liste. Program ListDemo čini to u metodu formulara FormDestroy:

```
procedure Tform1.FormDestroy (Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to ListDate.Count —1 do
    TObject (ListDate [I]).Free;
  ListDate.Free;
end;
```

Delphi 5 kontejner klase

Delphi 5 predstavlja novu seriju kontejnerskih klasa definisanih u jedinici Contnrs. Ove klase proširuju klase TList dodavanjem ideje o vlasništvu i definisanjem specifičnih pravila izdvajanja (imitirajući stekove i redove). Osnovna razlika između TList i nove klase TObjectList je da je ova poslednja definisana kao lista TObject objekata, ne kao lista pokazivača. Još je važnija činjenica da ukoliko objekat liste sadrži svojstvo OwnsObjects za koje je određena vrednost True, automatski se uklanja objekat kada ga zameni drugi objekat i uklanja se objekat kada se sama lista ukloni. Evo spiska svih novih kontejnerskih klasa:

- Klasa TobjectList, koju sam već opisao, predstavlja listu objekata koju poseduje sama lista.
- Izvedena klasa TComponentList predstavlja listu komponenata sa potpunom podrškom za notifikaciju uklanjanja (važna bezbednosna funkcija kada su dve komponente povezane svojim svojstvima, to jest, kada je komponenta vrednost svojstva druge komponente).
- Klasa TClassList je lista referenci klase. Izvedena je iz TList i ne zahteva uklanjanje.
- Klase TStack i TObjectStack predstavljaju liste pokzivača i objekata iz kojih možete samo izdvojiti elemente počevši od poslednjeg umetnutog objekta. Stek sledi LIFO poredak (poslednji uneti, prvi napolje — Last In, First Out). Tipični metodi za stek su Push za umetanje, Pop za izdvajanje i Peek za proveru prvog elementa bez uklanjanja elementa. Još uvek možete koristiti sve metode osnovne klase TList.

- Klase `TQueue` i `TObjectQueue` predstavljaju liste pokazivača i objekata iz kojih uvek izdvajate prvi uneti element (FIFO: prvi umetnuti, prvi napolje — First In, First Out). Metodi ovih klasa su jednaki metodima klasa steka, ali se drugačije ponašaju.

UPOZORENJE

Za razliku od klase `TObjectList`, klase `TObjectStack` i `TObjectQueue` ne poseduju umetnute objekte i neće ukloniti objekte koji ostaju u strukturi podataka kada je struktura uklonjena. Možete jednostavno izdvojiti sve elemente, ukloniti ih kada završite njihovo korišćenje, a zatim ukloniti kontejner. ■

Da bih demonstrirao upotrebu ovih klasa, ja sam modifikovao raniji primer `ListDate` u novi primer Contain. Prvo sam promenio tip promenljive `ListDate` u `TObjectList`. U metodu FormCreate sam modifikovao listu kreiranja u sledeći kod koji aktivira listu vlasništva:

```
ListDate := TObjectList.Create (True);
```

Sada možemo pojednostaviti kod uklanjanja jer će primena metoda `Free` nad listom automatski osloboditi datume koji se čuvaju.

Programu sam, takođe, dodao objekte steka i reda, popunjavajući svaki od njih brojevima. Jedna od dve kontrole formulara prikazuje listu brojeva u svakom od kontejnera, a druga uklanja poslednji element (prikazujući ga u dijalogu za poruke):

```
procedure TForm1.btnQueueClick(Sender TObject);
var
  I: Integer;
begin
  ListBox1.Clear;
  for I := 0 to Stack.Count - 1 do begin
    ListBox1.Items.Add (IntToStr (Integer (Queue.Peek)));
    Queue. Push(Queue.Pop);
end;
  ShowMessage ('Removed: ' + IntToStr (Integer (Stack.Pop)));
end;
```

Kada kliknete dve kontrole, možete videti kako poziv metoda Pop za svaki kontejner kao rezultat daje poslednji uneti element. Razlika je u tome što klasa `TQueue` umeće elemente na početak, a klasa `TStack` umeće elemente na kraj.

Sigurnost tipova kontejnera i lista

Kod kontejnera i lista postoji problem. Kod njih nema sigurnosti tipova kao što sam pokazao u oba primera dodavanjem objekta kontrole listi datuma. Da biste osigurali da podaci liste budu homogeni, možete proveriti tip podataka koje izdvajate pre nego što ih umetnete, ali dodatna mera bezbednosti je da proverite tip podataka prilikom izdvajanja. Ipak, dodavanje provere tipa u vreme izvršavanja usporava program, a i rizično je — programer možda neće moći da proveri tip za neke klase.

Da biste rešili oba problema, možete kreirati specifične klase lista za date tipove podataka i možete napisati kod u maniru klasa `TList` ili `TObjectList` (ili neke druge kontejnerske klase). Postoje dva pristupa kojima možete obaviti ovakav zadatak:

- Izvedite novu klasu iz klase liste i prilagodite metod `Add` i metode pristupa koji se odnose na svojstvo `Items`. To je metod koji koristi i Borland za kontejnerske klase, koje su sve izvedene iz klase `TList`.
- Kreirajte potpuno novu klasu koja sadrži objekat `TList` i mapirajte metode nove klase za internu listu koristeći pravilnu proveru tipova. Ovim pristupom definišete širu klasu, klasu koja obuhvata postojeću klasu da bi obezbedila različiti ili ograničeni pristup metodima (u našem slučaju, da bi izvršila konverziju tipova).

Ja sam implementirao oba rešenja u primer DateList, koji definiše liste `TDate` objekata. U narednom listingu ćete pronaći deklaraciju dveju klasa, izvedenu klasu `TDateListI` i širu klasu `TDateListW`.

```
type
// inheritance based
TDateListI = class (TObjectList)
protected
  procedure SetObject (Index: Integer; Item: TDate);
  function GetObject (Index: Integer): TDate;
public
  function Add (Obj: TDate): Integer;
  procedure Insert (Index: Integer; Obj: TDate);
  property Objects [Index: Integer]: TDate
    read GetObject write SetObject; default;
end;
// wrapper based
TDateListW = class(TObject)
private
  FList: TObjectList;
  function GetObject (Index: Integer): TDate;
  procedure SetObject (Index: Integer; Obj: TDate);
  function GetCount: Integer;
public
  constructor Create;
  destructor Destroy; override;
  function Add (Obj: TDate): Integer;
  function Remove (Obj: TDate): Integer:
  function IndexOf (Obj: TDate): Integer:
  property Count: Integer read GetCount:
  property Objects [Index: Integer]: TDate
    read GetObject write SetObject; default;
end;
```

Očigledno, prvu klasu je lakše napisati — sadrži manje metoda, a oni jednostavno pozivaju nasleđene metode. Dobra stvar je što se objekat `TDateListI` može proslediti parametrima koji očekuju `TList`. Problem je u tome što kod koji manipuliše instancom liste preko generičke promenljive `TList` neće pozivati specijalizovane metode jer oni nisu virtuelni i može se desiti dodavanje drugih tipova podataka listi objekata.

Ukoliko odlučite da ne koristite nasleđivanje, moraćete da napišete dosta koda jer je potrebno da reprodukujete svaki od originalnih metoda `TList` jednostavnim pozivanjem internog objekta `FList`. Nedostatak je što klasa `TDateListW` nije kompatibilna po tipu sa klasom `Tlist`, što ograničava njenu korisnost. Ne može se proslediti kao parametar metodima koji očekuju `TList`.

Oba pristupa obezbeđuju dobru proveru tipova podataka. Kada ste kreirali instancu jedne od ovih klasa, možete dodati samo objekte odgovrajućeg tipa podataka, a objekti koje izdvajate će naravno biti dobrog tipa podataka. Ovo je prikazano primerom DateList. Ovaj program sadrži nekoliko kontrola, kombo polje koje omogućava korisniku da izabere koju od lista će prikazati, i polje liste koje prikazuje vrednosti liste. Program razvlači listu pokušavajući da doda kontrolu listi `TDate` objekata. Da bi se dodao objekat drugačijeg tipa listi `TdateListI`, možemo jednostavno konvertovati listu u osnovnu klasu, klasu `TList`. Ovo se može slučajno dogoditi ukoliko prosledite listu kao parametar metodu koji očekuje objekat osnovne klase. Nasuprot tome, da bi lista `TDateListW` proizvela grešku, moramo eksplicitno konvertovati objekat u `TDate` pre umetanja u listu, ono što programer nikada ne treba da učini:

```
procedure TForm1.ButtonAddButtonClick (Sender: TObject);
begin
  ListW.Add (TDate (TButton.Create (nil)));
  TList(ListI).Add (TButton.Create (nil));
  UpdateList;
end;
```

Poziv `UpdateList` povlači izuzetak koji se prikazuje direktno u listi, jer sam ja koristio konverziju `as` za klase liste. Mudar programer neće nikada napisati prethodni kod.

Rezimirajmo: pisanje korisničke liste za specifični tip čini program robusnijim. Kreiranje šire liste umesto izvedene liste se čini nešto bezbednijim, mada zahteva više kodiranja.

NAPOMENA

Umesto da ponovo pišete šire klase liste za različite tipove, možete upotrebiti moj List Template Wizard, koji je predstavljen u knjizi "Delphi priručnik za programere" (Delphi Developer's Handbook), a koji možete pronaći na mom web sajtu. ■

Šta je sledeće?

Kao što smo videli u ovom poglavlju, Delphi sadrži veliki broj biblioteka klasa koje su velike koliko i Microsoftove biblioteke klasa MFC C++. Delphi VCL je, naravno, mnogo više orijentisan ka komponentama, i njegove klase nude viši nivo apstrakcije nad Windows API-jem nego što to obično čine biblioteke C++.

Da biste upotrebili komponente, potrebno je samo da potpuno razumete krajnje čvorove VCL hijerarhije, to jest, komponente koje se prikazuju u Component Palette i još nekoliko drugih komponenata. Zaista Vam nije potrebno šire znanje VCL interne strukture da biste upotrebili komponente; takvo znanje je potrebno samo ukoliko pišete nove komponente ili modifikujete postojeće.

Ovo poglavlje završava Deo I ove knjige, koji obuhvata osnove Delphi programiranja. Deo II je potpuno posvećen primerima upotrebe različitih komponenata. U Poglavlju 5 počećemo sa naprednom upotrebom tradicionalnih Windows kontrola i menija, u Poglavlju 6 ćemo se pozabaviti klasom `TForm`, a zatim ćemo se pozabaviti paletama alata, statusnim linijama, okvirima za dijalog i MDI aplikacijama u kasnijim poglavljima.

DEO

II

Upotreba komponenata

U OVOM DELU:

POGLAVLJE

5

Napredna upotreba standardnih komponenata

SADA, KADA STE UPOZNALI DELPHI OKRUŽENJE I KADA STE PROČITALI UVOD U JEZIK OBJECT PASCAL I VISUAL COMPONENT LIBRARY, SPREMNI SMO DA ISTRAŽIMO DRUGI DEO KNJIGE: UPOTREBU KOMPONENATA. TO JE ONO ŠTO JE DELPHI ZAPRAVO. VIZUELNO PROGRAMIRANJE UPOTREBOM KOMPONENATA JE KLJUČNA KARAKTERISTIKA OVOG RAZVOJNOG OKRUŽENJA.

Delphi dobijate sa velikim brojem gotovih komponenata. Ja neću detaljno opisivati svaku komponentu, razmatrajući svako od njenih svojstava ili svaki od metoda. Ukoliko Vam je potrebna ovakva informacija, možete je lako pronaći u Help sistemu. Cilj Dela II ove knjige je da Vam pokaže kako da upotrebite neke od naprednijih karakteristika koje nude unapred određene komponente Delphija za izradu aplikacija.

Ja ću početi navođenjem svih različitih alternativa koje su Vam na raspolaganju, jer izbor prave komponente često znači i brži razvoj projekta. Ovo poglavlje predstavlja komponente strane Standard sa Component Palette i neke Win32 kontrole.

Otvaranje palete alata Component

Dakle, želite da napišete Delphi aplikaciju. Otvarate novi projekat u Delphiju i ispred Vas se pojavljuje veliki broj komponenata. Problem je što za svaku operaciju postoji više alternativa. Na primer, možete prikazati listu vrednosti koristeći listu, kombo, radio grupu, tabelu stringova ili čak drvo ukoliko postoji hijerarhijski poredak. Koju komponentu treba upotrebiti? To je teško reći. Treba uzeti u obzir mnoge stvari, već prema onome što imate na umu da bi Vaša aplikacija trebalo da radi. Zbog toga sam pripremio poprilično skraćeni rezime alternativnih mogućnosti za nekoliko uobičajenih zadataka.

NAPOMENA

Za neke kontrole, koje će biti opisane u narednim odeljcima, Delphi takođe sadrži verziju koja prepoznaje podatke, obično označenu prefiksom DB. Kao što ćete videti u Poglavlju 9, DB verzija kontrole obično služi kao i njen "standardni" ekvivalent, ali svojstva i načini upotrebe su često potpuno drugačiji. Na primer, za kontrolu Edit koristite svojstvo `Text`, dok za komponentu DBEdit pristupate svojstvu `Value` odgovarajućeg polja. ■

Komponente za unos teksta

Mada komponenta ili formular mogu direktno obraditi unos sa tastature koristeći događaj `OnKeyPress`, to nije uobičajena operacija. Windows obezbeđuje kontrole koje možete upotrebiti za prihvatanje stringa pa čak i za izradu jednostavnog editora teksta. Delphi sadrži nekoliko drugačijih komponenata za ovu namenu.

Komponenta Edit

Komponenta Edit omogućava korisniku da unese jednu liniju teksta. (Možete, takođe, prikazati jednu liniju teksta upotrebom kontrola Label i StaticText, ali se komponente obično koriste za nepromenljivi tekst ili izlaz koji generiše program, ne za unos.) Komponenta Edit koristi svojstvo `Text` dok mnoge druge kontrole koriste svojstvo `Caption` da bi se referisale na tekst koji prikazuju. Jedini uslov koji možete da postavite korisniku prilikom unosa je broj karaktera koji se može prihvatiti. Ukoliko želite da prihvatite samo određene karaktere, možete obraditi događaj `OnKeyPress` polja za izmene. Na primer, možemo napisati metod koji testira da li je karakter cifra ili taster Backspace (čija je numerička vrednost 8). Ukoliko nije, možemo promeniti vrednost karaktera u null karakter (#0) tako da se neće obraditi kontrolom za izmene i proizvešće zvuk upozorenja:

```
procedure TForm1.EditKeyPres (
  Sender: TObject; var Key: Char);
begin
  // check if the key is a number or backspace
  if not (Key in ['0' .. '9', #8]) then
  begin
    Key := 0;
    Beep;
  end;
end;
```

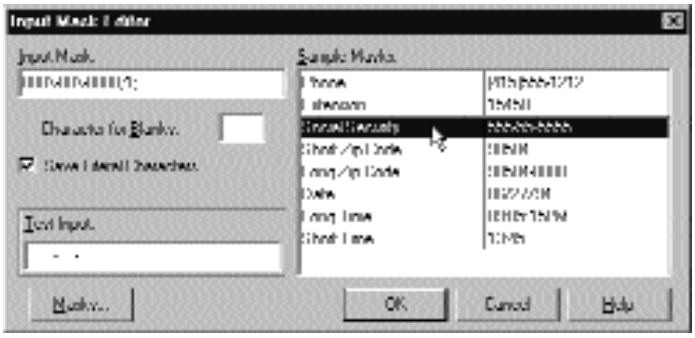
Komponenta MaskEdit

Da biste još više prilagodili ulaz polja za izmene, možete upotrebiti komponentu MaskEdit koja ima svojstvo `MaskEdit`. To je string koji naznačava da li karakter treba da bude prikazan malim slovima, velikim slovima ili ciframa, a možete navesti i druge slične uslove. Na slici 5.1 možete videti editor svojstva `MaskEdit`.

SAVET

Možete prikazati editor bilo kog svojstva izborom svojstva u Object Inspectoru i kada kliknete na tri tačke (...). ■

Editor InputMask Vam omogućava da unesete masku, ali, takođe, zahteva da naznačite karakter koji treba koristiti za obeležavanje mesta karaktera pri unosu i da odredite da li treba sačuvati literale (literals) koji predstavljaju masku uz konačni string. Na primer, možete odabrati zagrade oko pozivnog broja telefona samo kao pomoć pri unosu ili ih možete sačuvati uz string koji sadrži rezultujući broj. Ova dva elementa u editoru Mask odgovaraju dvama poslednjim poljima maske (odvojena su tačkom i zarezom).



SLIKA 5.1 *Editor svojstva EditMask komponente MaskEdit*

SAVET

Pritiskanje kontrole Mask u Mask Editoru omogućiće Vam da odaberete unapred određene ulazne maske za različite zemlje. ■

Komponente Memo i RichEdit

Obe kontrole koje smo do sada razmatrali su dozvoljavale samo po jednu liniju sa ulaza. Komponenta Memo, nasuprot tome, može prihvatiti nekoliko linija teksta (na platformama Win 95/98), ali zadržava limit od 32KB sa 16-bitnih Windowsa i omogućava samo jedan font za ceo tekst. Sa tekstom možete raditi liniju po liniju (koristeći listu stringova `Lines`) ili možete pristupiti celom tekstu odjednom (koristeći svojstvo `Text`).

Ukoliko želite da prihvatite veliku količinu teksta ili promenite fontove i poravnanje pasusa, trebalo bi da koristite kontrolu RichText, Win32 kontrolu koja se zasniva na RTF formatu dokumenta. Možete pronaći primer kompletnog editora zasnovanog na komponenti RichText među primerima programa koje dobijate uz Delphi. (Primer je nazvan RichText.)

Za komponentu RichText postoji svojstvo `DefAttributes` koje naznačava unapred određene stilove, i svojstvo `SelAttributes` koje naznačava stil aktuelne selekcije. Ova dva svojstva nisu tipa `Tfont`, ali su kompatibilna sa fontovima tako da možemo koristiti metod `Assign` da bismo kopirali vrednost, što je pokazano narednim fragmentom koda:

```
procedure TForm1.ButtonClick (Sender: TObject);
begin
  if RichEdit1.SelLength > 0 then
  begin
    FontDialog1.Font.Assign (RichEdit1.DefAttributes);
    if FontDialog1.Execute then
      RichEdit1.AelAttributes.Assign (FontDialog1.Font);
  end;
end;
```

Odabiranje opcija

Postoje dve standardne Windows kontrole koje omogućavaju korisniku da odabere različite opcije, kao i kontrole za grupisanje skupova opcija.

Komponente CheckBox i RadioButton

Prva komponenta je polje za potvrdu (check box) koje odgovara opciji koja se može odabrati bez obzira na status ostalih polja za potvrdu. Dodeljivanje vrednosti svojstvu `AllowGrayed` polja za potvrdu Vam omogućava da prikažete tri različita stanja (selektovano, nije selektovano i sivo), koja se menjaju kada korisnik klikne polja za potvrdu.

Drugi tip kontrole je opciona kontrola (radio button) koja odgovara ekskluzivnom izboru. Dve opcione kontrole u okviru jednog formulara ili u okviru grupe opcija kontejnera ne mogu biti istovremeno selektovane, a jedna od njih uvek treba da bude selektovana (kao programer, Vi ste odgovorni za selektovanje jedne od opcionih kontrola u vreme dizajniranja).

Komponente GroupBox

Da biste prihvatili nekoliko grupa opcionih kontrola, možete upotrebiti GroupBox koji će sadržati sve kontrole, kako funkcionalno tako i vizuelno. Da biste kreirali GroupBox sa opcionim kontrolama, jednostavno postavite komponentu GroupBox na formular, a zatim dodajte opcione kontrole na GroupBox.

Opcione kontrole možete obraditi pojedinačno, ali je lakše kretati se kroz niz kontrola koje poseduje GroupBox, kao što je opisano u prethodnom poglavlju. Evo malog isečka koda koji je upotrebljen za dobijanje teksta selektovane opcione kontrole:

```
var
  I: Integer;
  Text: string;
begin
  for I := 0 to GroupBox1.ControlCount —1 do
    if (GroupBox1.Controls[I] as TRadioButton).Checked then
      Text := (GroupBox1.Controls[I] as TRadioButton).Caption;
```

Komponenta RadioGroup

Delphi sadrži sličnu komponentu koja se može upotrebiti za opcione kontrole, komponentu RadioGroup. RadioGroup je polje grupe sa nekoliko *klonova* opcionih kontrola. Termin *klon* u ovom kontekstu se odnosi na činjenicu da je komponenta RadioGroup jedna kontrola, jedan prozor, sa elementima sličnim opcionim kontrolama koje su prikazane na njoj.

Upotreba komponente RadioGroup je obično jednostavnija od upotrebe komponente GroupBox, jer su različiti elementi deo liste, kao kod polja List. Ovako možete dobiti tekst selektovanog elementa:

```
Text := RadioGroup1.Items [RadioGroup1.ItemIndex];
```

RadioGroup koristi manje resursa i manje memorije, i trebalo bi da bude lakše kreirati je i ponovo iscrtati. Takođe, komponenta RadioGroup može automatski poravnati svoje opcione kontrole u jednoj ili više kolona (što je naznačeno svojstvom `Columns`) i lako možete dodati nove elemente u vreme izvršavanja, dodavanjem stringova listi stringova `Items`. Nasuprot tome, dodavanje novih opcionih kontrola u GroupBox bi bilo prilično složeno.

Liste

Kada imate mnogo opcija, opcione kontrole nisu najbolje rešenje. Uobičajeno je da broj opcionih kontrola ne prelazi pet ili šest opcionih kontrola, da bi se izbeglo pretrpavanje korisničkog interfejsa; kada imate više opcija, možete upotrebiti listu ili neku od drugih kontrola koje prikazuju listu elemenata i omogućavaju selektovanje nekog od njih.

Komponenta ListBox

Selekcija elementa u listi koristi svojstva `Items` i `ItemIndex` kao u kodu prikazanom za RadioGroup. Ukoliko je potrebno da pristupite tekstu selektovanih elemenata liste, često možete napisati širu funkciju kakva je ova:

```
function SelText (List: TListBox): string;
var
  nItem: Integer;
begin
  nItem := List.ItemIndex;
  if nItem >=0 then
    Result := List.Items [nItem]
  else
    Result := '';
end;
```

Još jedna važna karakteristika je što upotrebom ListBoxa možete odabrati da li da dozvolite pojedinačano selektovanje, kao kod grupe opcionih kontrola, ili da dozvolite višestruko selektovanje, kao kod grupe polja za potvrdu. Određivanjem vrednosti svojstva `MultiSelect` određujete jedno od ova dva ponašanja. Postoje dve vrste višestrukih selekcija u Windowsu i u Delphi listama: višestruka selekcija (multiple selection) i proširena selekcija (extended selection). U prvom slučaju korisnik selektuje više elemenata tako što klikne elemente, dok u drugom slučaju korisnik može upotrebiti tastere Shift i Ctrl da bi selektovao više uzastopnih elemenata ili elemente koji nisu uzastopni. Druga mogućnost se određuje svojstvom `ExtendedSelect`.

Za listu sa više selektovanih elemenata program može pročitati koliko je elemenata selektovano upotrebom svojstva `SelCount`, i može odrediti koji elementi su selektovani proverom niza `Selected`. Ovaj niz `Boolean` vrednosti ima jednak broj elemenata kao i lista. Na primer, da biste povezali sve selektovane elemente u jedan string, možete pretražiti niz `Selected` na sledeći način:

```
var
  SelItems: string;
  nItem: Integer;
begin
  SelItems := '';
  for nItem := 0 to ListBox1.Items.Count – 1 do
    if ListBox1.Selected [nItem] then
      SelItems := SelItems + ListBox1.Items[nItem] + ' ';
```

Komponenta ComboBox

Liste zauzimaju dosta ekranskog prostora i omogućavaju fiksnu selekciju. To znači da korisnik može odabrati samo element koji postoji u listi i ne može uneti element koji programer nije predvideo.

Oba problema možete rešiti upotrebom kontrole ComboBox, koja predstavlja kombinaciju polja za izmene i liste. Ponašanje komponente ComboBox se umnogome menja u zavisnosti od vrednosti svojstva `Style`. Stil `csDropDown` definiše tipični ComboBox koji omogućava direktno editovanje i prikazuje listu na zahtev korisnika, stil `csDropDownList` definiše ComboBox koji ne omogućava editovanje (ali koristi ulaz sa tastature za selektovanje elementa), a stil `csSimple` definiše ComboBox koji uvek prikazuje listu.

Pristupanje teksta selektovane vrednosti ComboBoxa je lakše od iste operacije za ListBox, jer možete jednostavno upotrebiti svojstvo `Text`. Koristan i uobičajen trik za ComboBox je dodavanje novog elementa listi kada korisnik unese tekst i pritisne taster Enter. Naredni metod prvo testira da li je korisnik pritisnuo taj taster, proverom karaktera sa numeričkom vrednošću (ASCII) 13. Zatim proverava da ComboBox nije prazan i da li vrednost već postoji u listi — da li je njegova pozicija manja od nule. Evo koda:

```
procedure TForm1.ComboBox1KeyPress (
  Sender: TObject; var Key: Char);
begin
  // if the user presses the Enter key
  if Key = chr (13) then
    with ComboBox3 do
      if (Text <> '') and (Items.IndexOf (Text) < 0) then
        Items.Add (Text);
end;
```

Komponenta CheckListBox

Još jedno proširenje kontrole ListBox predstavlja komponenta CheckListBox, lista u kojoj je svaki element predstavljen poljem za potvrdu (kao što možete videti na slici 5.2). Korisnik može selektovati jedan element liste, ali može i kliknuti na elemente da bi promenio stanje polja za potvrdu. Ovo čini CheckListBox veoma dobrom komponentom za višestruke selekcije ili za isticanje statusa serije nezavisnih elemenata (kao kod serije polja za potvrdu).

SLIKA 5.2 *Korisnički interfejs kontrole CheckListBox, koja je u osnovi lista polja za potvrdu*

Da biste proverili trenutni status svakog od elemenata, možete upotrebiti nizove svojstava `Checked` i `State` (upotrebite ovaj drugi niz ukoliko polja za potvrdu mogu biti siva). Delphi 5 predstavlja niz svojstava `ItemEnabled`, koje možete upotrebiti da aktivirate ili deaktivirate svaki od elemenata liste. Upotrebićemo CheckListBox u primeru DragList kasnije u ovom poglavlju.

SAVET

Većina kontrola na osnovu lista ima važnu zajedničku karakteristiku. Svakom elementu liste je pridružena 32-bitna vrednost, obično naznačena tipom `TObject`. Ova vrednost se može upotrebiti kao tag za svaki element liste i veoma je korisna za čuvanje dodatnih informacija uz element. Ovaj pristup ima veze sa specifičnom karakteristikom Windowsa koja nudi četiri bajta dodatnog prostora za čuvanje svakog elementa liste. Ovu karakteristiku ćemo upotrebiti u primeru ODList kasnije u ovom poglavlju. ■

Komponente ListView i TreeView

Ukoliko želite još sofisticiraniju listu, možete upotrebiti Win32 kontrolu ListView koja će učiniti da Vaš korisnički interfejs izgleda još savremenije. Ova komponenta je nešto složenija za upotrebu, a opisaćemo je pri kraju ovog poglavlja. Preostale alternative prikazivanja lista su kontrola TreeView koja prikazuje elemente u hijerarhijskom izlazu, i kontrola StringGrid koja u svakoj liniji prikazuje više elemenata. Kontrola StringGrid je opisana u dodatnom poglavlju "Grafika u Delphiju" koje možete pronaći na adresi `www.sybex.com`.

Ukoliko koristite uobičajene kontrole u Vašoj aplikaciji, korisnici će već znati kako da ih upotrebe i smatraće korisnički interfejs Vaše aplikacije savremenim. TreeView i ListView su dve ključne komponente Windows Explorera i možete pretpostaviti da ih poznaju mnogi korisnici, čak i da su više upoznati sa ovim kontrolama nego sa tradicionalnim Windows kontrolama.

Opsezi

Konačno postoji nekoliko komponenata koje možete upotrebiti za selektovanje vrednosti iz određenog opsega. Opsezi se mogu koristiti za numerički unos i za selektovanje elementa iz liste.

Komponenta ScrollBar

Samostalna kontrola ScrollBar je originalna komponenta ove grupe, ali se često koristi zasebno. ScrollBarovi se obično koriste uz ostale komponente kao što su liste i Memo polja, ili se koriste uz formulare. U svim ovim slučajevima ScrollBar se može smatrati delom površine neke druge komponente. Na primer, formular sa ScrollBarom je zapravo formular koji podseća na formular na kojem je ScrollBar iscrtan pri njegovoj granici, karakteristika kojom upravlja specifični Windows stil prozora formulara. Pod terminom *podseća* sam podrazumevao da se ScrollBar ne nalazi u zasebnom prozoru. Ovi lažni ScrollBarovi se obično u Delphiju kontrolišu upotrebom specifičnih svojstava formulara i drugih komponenata na kojima se nalaze.

Komponente TrackBar i ProgressBar

Direktna upotreba kontrole ScrollBar je veoma retka, naročito od kada je predstavljena komponenta TrackBar u Windowsu 95, koja se koristi da bi se korisniku omogućilo selektovanje vrednosti iz opsega. Među uobičajenim kontrolama Win32 nalazi se i kontrola ProgressBar koja omogućava programu da izda vrednost iz opsega, prikazujući tako napredak dugotrajnih operacija.

Komponenta UpDown

Još jedna od ovih kontrola je kontrola UpDown koja je obično povezana sa poljem za izmene, tako da korisnik može uneti broj ili povećati ili smanjiti broj koristeći dve male kontrole koje su označene strelicama. Ništa Vas ne sprečava da komponentu UpDown koristite samostalno, prikazujući trenutnu vrednost u oznaci, ili na neki drugi način.

Komponenta PageScroller

Win32 kontrola PageScroller je kontejner koji Vam omogućava da prolazite kroz interne kontrole. Na primer, ukoliko na PageScroller postavite paletu alata, a paleta alata je veća od mogućeg prostora, PageScroller će prikazati dve male strelice na svakoj strani. Kada kliknete na ove strelice, prolazićete kroz internu površinu. Ova komponenta se može koristiti kao klizač, ali takođe praktično zamenjuje kontrolu ScrollBox.

Komponenta ScrollBox

Kontrola ScrollBox predstavlja oblast formulara kroz koju možete prolaziti nezavisno od ostatka površine formulara. Zbog toga ScrollBox sadrži dva klizača koja se koriste za pomeranje internih komponenata. Lako možete postaviti druge komponente unutar ScrollBoxa, kao što biste to učinili sa panelom. Zapravo, ScrollBox je u osnovi panel sa klizačima kojima se pomera interna površina, što je interfejs koji se koristi u mnogim Windows aplikacijama. Kada imate formular sa mnogo kontrola i paletom alata ili statusnom linijom, možete upotrebiti ScrollBox da biste prekrili centralnu površinu formulara, ostavljajući klizače i statusne linije van oblasti. Oslanjajući se na klizače formulara, zapravo, možete omogućiti korisniku da pomeri paletu alata ili statusnu liniju van oblasti za prikazivanje, što je veoma čudna situacija.

Prevlačenje sa jedne komponente na drugu

Sada kada ste se upoznali sa standardnim kontrolama, razmotrićemo par opštih tehnika: obradu prevlačenja i fokusiranja. Dozvolite mi da počnem jednostavnim primerom prevlačenja, nazvanim DragList. Formular ovog primera, prikazn na slici 5.3 u vreme izvršavanja, sadrži ListBox i CheckListBox. Možete prevlačiti elemente sa jedne kontrole na drugu kontrolu. Takođe, sadrži polje za izmene koje možete upotrebiti za unos novih elemenata i da ih prevučete na jednu od lista. Ukoliko pokrenete program, videćete da postoji pravilo: liste ne mogu sadržati više istih elemenata. To znači da moramo proveriti da li element već postoji u listi pre nego što ga umetnemo u listu.

SLIKA 5.3 *Formular primera DragList u vreme izvršavanja prilikom operacije prevlačenja*

Dve liste koriste vrednost `dmAutomatic` svojstva `DragMode` (uz svojstvo `DragKind` za koje je ostavljena unapred određena vrednost `dkDrag`). Za polje za izmene, nasuprot tome, moramo upotrebiti ručno prevlačenje da bismo omogućili polju za izmene da se uobičajeno ponaša kada korisnik klikne polje. Zbog toga, kada korisnik klikne taster miša iznad polja za izmene, mi moramo inicirati operaciju prevlačenja, usporavajući je kako je naznačeno prvim parametrom metoda `BeginDrag`:

```
procedure TDragForm.Edit1MouseDown (Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  Edit1.BegirnDrag (False, 10);
end;
```

Dve liste dele istu obradu događaja `OnDragOver` koja se koristi da bi se odredilo da li kontrola prihvata prevlačenje sa datog izvora. U ovoj rutini kada korisnik prevlači sa polja za izmene, program proverava da li tekst već postoji u listi i onemogućava prevlačenje ukoliko postoji. Lako možemo napisati jednu rutinu za obradu događaja za obe kontrole jer su kontrole izvedene iz iste osnovne klase, klase `TCustomListBox`:

```
procedure TDragForm.ListDragOver (Sender.Source: TObject;
  X, Y: Integer; State: TDragState; var Accept: Boolean);
begin
  Accept := True;
  // if the source is the edit and the items
  // is already in the destination list, reject it
if (Source = Edit1) and
    ((Sender as TCustoeListBox).rtems.IndexOf (Editl.Text) >= 0) then
    Accept := False;
end;
```

Rutine događaja `OnDragDrop` su ipak prilično različite pa sam odlučio da ih razdvojim. Lista omogućava da se selektuje samo jedan element, dok lista polja za potvrdu omogućava selektovanje više elemenata liste; to čini kod prilično različitim u ova dva slučaja. Ono što može biti zajedničko u ovom kodu je dodavanje elementa listi ukoliko element ne postoji. Ja sm dodao zajednički kod metodu formulara koji pozivaju obe rutine za obradu događaja:

```
function TDragForm.AddNotDup (List: TCustomListBox;
  Text: string): Boolean;
begin
  // return if the string was not already in the list
  Result := List.Items.IndexOf (Text) < 0;
  if Result then
    List. Items. Add (Text);
end;
```

Kod za dva metoda prevlačenja je prilično jednostavan. Za listu polja za potvrdu program kopira tekst iz polja za izmene ili selektovanog elementa liste i uklanja ga iz izvora:

```
procedure TDragForm.checkListBox1DragDrop(
  Sender, Source: T0bject; X, Y: Integer);
var
  nItem: Integer;
begin
  if Source = Edit1 then
    // copy the text of the edit box
    CheckListBoxl. Items Add (Edit1.Text)
  else if Source = ListBox1 then
  begin
    // copy if not duplicate
    nItem := ListBox1.ItemIndex;
    if AddNotDup (CheckListsBox1, ListBoxl.Items [nItem]) then
      // remove source item
      ListBox1.Items.Delete (nItem);
  end;
end;
```

Za listu moramo proveriti sve elemente liste polja za potvrdu da bismo videli koji element je selektovan. Pošto želimo da uklonimo elemente koje kopiramo, ovu operaciju moramo izvršiti u obrnutom redosledu, jer uklanjanje elemenata menja položaj elemenata koji slede element koji uklanjamo:

```
procedure TDragForm.ListBax1DragDrop(Sender,
  Source: TObject; X, Y: Integer);
var
  I: Integer;
begin
  if Source = Edit1 then
    // copy the text of the edit box
    ListBox1.Items.Add (Edit1.Text)
  else if Source = CheckListBox1 then
  begin
    // copy all the selected items (unless duplicate)
    // and delete them (using reverse order!)
    for I := CheckListBox1.Items.Count - 1 downto 0 do
      if CheckListBox1.Checked [I] then
      begin
        if AddNotDup (ListBox1, CheckListBox1.Items [I]) then
          CheckListBox1.Items.Delete (I);
      end;
  end;
end;
```

NAPOMENA

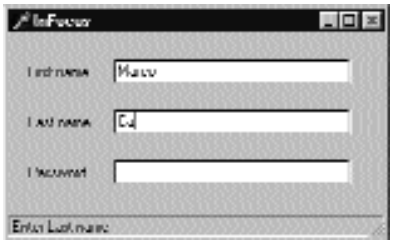
Primer operacija prevlačenja ćemo videti u okviru kontrole TreeView na kraju ovog poglavlja. ■

Obrada ulaznog fokusa

Upotrebom svojstava `TabStop` i `TabOrder` koja postoje za većinu kontrola možete odrediti redosled po kome kontrole dobijaju ulazni fokus kada korisnik pritisne taster Tab. Umesto da ručno podesite svojstvo TabOrder za svaku od komponenata formulara, možete upotrebiti iskačući meni Form Designera da biste aktivirali okvir za dijalog Edit Tab Order, kao što je prikazano na slici 5.4.

Pored ovih osnovnih svojstava, veoma je važno znati da svaki put kada komponenta dobije ili izgubi ulazni fokus, ona dobija i odgovarajući događaj `OnEnter` ili `OnExit`. Ovo Vam omogućava da fino podesite i prilagodite redosled korisnikovih operacija. Neke od ovih tehnika su prikazane u primeru InFocus koji kreira prilično uobičajeni prozor prijave korisnika. Formular primera sadrži tri polja za izmene sa oznakama koje objašnjavaju njihovo značenje, kao što je prikazano na slici 5.5. Pri dnu prozora se nalazi statusna oblast koja daje obaveštenja koja vode korisnika. Svaki od elemenata je potrebno uneti po određenom redosledu.

SLIKA 5.4 *Okvir za dijalog Edit Tab Order*



SLIKA 5.5 *Primer InFocus u vreme izvršavanja*

Za izlaz statusne informacije ja sam koristio komponentu StatusBar koja predstavlja jedno izlazno polje (što se postiže određivanjem vrednosti `True` za svojstvo `SimplePanel`). Evo rezimea svojstava ovog primera. Obratite pažnju na karakter `&` u oznakama, koji naznačava tastaturnu prečicu, i veze ovih oznaka sa odgovarajućim poljima za izmene (koristi se svojstvo `FocusControl`):

```
object FocusFonm: TFocusForm
  ActiveControl = EditFirstName
  Caption = 'InFocus'
  object Label1: TLabel
    Caption = '&Firat name'
    FocusControl = EditFirstName
  end
  object EditFirstName: TEdit
    OnEnter = GlobalEnter
    OnExit = EditFirstNameExit
  end
  object Label2: TLabel
    Caption = '&Last name'
    FocusControl = EditLastName
  end
  object EditLastName: TEdit
    OnEnter = GlobalEnter
  end
```

```
  object Label3: TLabel
    Caption = '&Password'
    FocusControl = EditPassword
  end
  object EditPassword: TEdit
    PasswordChar = '*'
    OnEnter = GlobalEnter
  end
  object StatusBar1: TStatusBar
    SimplePanel = True
  end
end
```

Program je veoma jednostavan i obavlja samo dve operacije. Prva je identifikacija, na statusnoj liniji, kontrole za izmene koja ima fokus. To se postiže obradom događaja `OnEnter`, sa mogućom upotrebom generičke obrade događaja da bi se izbeglo ponavljanje koda. U primeru sam, umesto čuvanja dodatnih informacija za svako polje za izmene, proverio svaku kontrolu formulara da bih odredio koja oznaka je dodeljena aktuelnom polju za izmene (naznačeno parametrom `Sender`):

```
procedure TFocusForm.GlobalEnter(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to ControlCount - 1 do
    // if the control is a label
    if (Controls [I] is TLabel) and
      // and the label is connected to the current edit box
      (TLabel(Controls[I]).FocusControl = Sender) then
    // copy the text leaving off the initial & character
    StatusBar1.SimpleText := 'Enter' +
      Copy (TLabel(Controls[I]).Caption. 2, 1000);
end;
```

Druga rutina za obradu događaja formulara se odnosi na događaj `OnExit` prvog polja za izmene. Ukoliko je kontrola ostala prazna, odbija se oslobađanje fokusa i vraća se na polje pre nego što se prikaže poruka korisniku. Metodi, takođe, proveravaju datu vrednost automatski popunjavajući drugo polje za izmene i premeštajući fokus direktno na treće polje:

```
procedure TFocusForm.EditFirstNameExit(Sender TObject);
begin
  if EditFirstName.Text = '' then
  begin
    // don't let the user get out
    EditFirstName.SetFocus;
    MessageDlg ('First name is required',
      mtError, [mbOK], 0);
  end
  else if EditFirstName.Text = 'Admin' then
  begin
    // fill the second edit and jump to the third
    EditLasthame.Text := 'Adamin'
    EditPassword. SetFocus;
  end;
end;
```

Rad sa menijima

Rad sa menijima i elementima menija je prilično jednostavan. Ovaj odeljak nudi samo kratke beleške i nekoliko naprednijih primera. Prva stvar koju treba imati na umu, a tiče se elemenata menija, je da mogu poslužiti različitim namenama:

- **KOMANDE** (commands) su elementi menija koji izvršavaju akciju.
- **ODREĐIVAČI STANJA** (state-setters) su elementi menija koji se koriste za ključivanje ili isključivanje opcije, za promenu stanja određenog elementa. Komande, obično sa leve strane, imaju znak za potvrdu čime je naznačeno da su aktivne.
- **OPCIONI ELEMENTI** (radio items) su označeni okruglim znakom za potvrdu i grupisani su da bi predstavili alternativne selekcije. Da biste dobili opcione elemente, jednostavno za svojstvo `RadioItem` odredite vrednost `True`, a istu vrednost odredite i za svojstvo `GroupIndex` alternativnih elemenata menija.
- **ELEMENTI MENIJA DIJALOGA** (dialog menu items) povlače prikazivanje okvira za dijalog i obično su označeni trima tačkama (. . .) iza teksta elementa.

Kada unosite nove elemente u Menu Designer, Delphi kreira novu komponentu za svaki element menija i prikazuje ga u Object Inspectoru (mada ništa nije dodato formularu). Da bi dodelio ime, Delphi koristi naslov koji unesete i dodaje mu broj (tako da *Open* postaje `Open1`). Pošto Delphi uklanja razmake i druge specijalne karaktere iz naslova prilikom kreiranja naziva, a separatori menija se određuju upotrebom crtice, ovi elementi neće imati naziv. Zbog toga Delphi dodaje slovo N nazivu kojem dodeljuje broj i generiše elemente `N1`, `N2` i tako dalje.

UPOZORENJE

Nemojte koristiti svojstvo Break koje se koristi za uređenje menija sa više kolona. Vrednost `mbMenuBarBreak` označava da će se element prikazati u drugoj ili narednoj liniji, a vrednost `mbMenuBreak` da će element biti dodat drugoj ili narednoj koloni menija. ■

Prečice u Delphiju 5

U Delphiju 5 ne morate uneti karakter `&` za `Caption` elementa menija; Delphi automatski obezbeđuje prečicu ukoliko je izostavite. Delphi 5 automatski sistem prečica može, takođe, pretpostaviti da li ste uneli prečicu koja već postoji i popraviti je. To ne znači da treba da prestanete da dodajete svoje prečice upotrebom karkaktera `&`, jer automatski sistem prečica koristi prvo slobodno slovo i ne sledi unapred određene standarde. Takođe, možete odrediti bolje mnemoničke tastere nego što bi to učinio automatski sistem.

Ova nova karakteristika Delphija 5 se kontroliše svojstvom `AutoHotkeys` koje je dostupno iz glavne komponente i u svakom od menija i elemenata menija. U glavnom meniju za ovo svojsvo je unapred određena vrednost `maAutomatic`, dok je za menije i elemente menija unapred određena vrednost `maParent` tako da se vrednost koju ste odredili za glavni meni automatski koristi i za podelemente, izuzev ukoliko za njih ne odredite vrednost `maAutomatic` ili `maManual`.

Mehanizam iza sistema je svojstvo `RethinkHotkeys` metod klase `TMenuItem` i pridruženi `InternalRethinkHotkeys`. Postoji i metod `RethinkLines` koji proverava da li meni sadrži dva

uzastopna separatora, ili počinje ili se završava separatorom. U svim ovim slučajevima separator se uklanja.

Jedan od razloga zašto Delphi sadrži ovu novu karakteristiku je novi ITE (Integrated Translation Evironment — integrisano okruženje prevođenja). Kada je potrebno da prevedete meni aplikacije, velika je pomoć ako ne morate da se bavite prečicama, ili da bar ne morate brinuti da li dva elementa menija imaju istu prečicu. Postojanje sistema koji može rešiti slične probleme je velika prednost. Drugi razlog je sam Delphi IDE. Sa svim dinamički učitanim paketima koji instaliraju menije u glavni IDE meni ili iskačuće menije, i sa različitim paketima koji se učitavaju u različite verzije proizvoda, gotovo je nemoguće načiniti prečice koje se ne ponavljaju. To je razlog zbog kojeg mehanizam nije načinjen kao čarobnjak koji vrši statičku analizu Vaših menija u vreme dizajniranja; načinjen je da rešava realne probleme manipulisanja menijima koji se dinamički kreiraju u vreme izvršavanja.

UPOZORENJE

Ova nova karakteristika je svakako veoma zgodna, ali pošto je unapred određeno da je aktivna, može pokvariti postojeći kod. Ja sam morao da modifikujem dva primera programa ovog poglavlja iz prethodnog izdanja knjige da bih izbegao greške prilikom izvršavanja koje bi prouzrokovala ova izmena. Kao što ćete kasnije videti, problem je što ja koristim naslov u okviru koda, a dodatni karakter & je kvario moj kod. Izmena je veoma jednostavna jer sve što je trebalo da učinim je da za svojstvo `AutoHotKeys` odaberem vrednost `maManual`. ■

Iskačući meniji i događaj OnContextPopup

Pored komponente MainMenu možete koristiti i komponentu PopupMenu. Ova komponenta se obično prikazuje kada korisnik klikne desnim tasterom miša komponentu koja koristi dati iskačući meni kao vrednost događaja `PopupMenu`.

Pored povezivanja iskačućeg menija sa komponentom preko odgovarajućeg svojstva, Vi možete pozvati i metod `Popup` za koji su potrebne ekranske koordinate iskačućeg menija. Pravilne vrednosti se mogu dobiti konvertovanjem lokalne tačke u ekransku tačku metodom `ClientToScreen` lokalne komponente, a to je u ovom isečku koda oznaka:

```
procedure TForm1.Label3MouseDown(Sender TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
var
  ScreenPoint: TPoint;
begin
  // if some condition applies...
  if Button = mbRight then
  begin
    ScreenPoint := Label3.ClientToScreen (
      Point (X, Y));
    PopupMenu1.Popup (ScreenPoint.X, ScreenPoint.Y)
  end;
end;
```

Drugačiji pristup koji obezbeđuje Delphi 5 je upotreba događaja `OnContextMenu`. Ovaj potpuno novi događaj se inicijalizuje kada korisnik klikne komponentu desnim tasterom miša, a to je tačno ono što smo pratili testom `if Button = mbRight`. Prednost je što se isti događaj poziva i

kada se pritisne kombinacija tastera Shift+F10, kao i bilo koji način korisničkog ulaza koji je definisan opcijama Windows Accessibility i hardverom (uključujući tastaturne prečice nekih Windows kompatibilnih tastatura). Možemo upotrebiti ovaj događaj da bismo prikazali iskačući meni uz malo kodiranja:

```
procedure TFormPopup.Label1ContextPopup(Sender TObject;
  MousePos: TPoint; var Handled: Boolean);
var
  ScreenPoint: TPoint;
begin
  // add dynamic items
  PopupMenu2.Items.Add (NewLine);
  PopupMenu2.Items.Add (NewItem (TimeToStr (Now),
    0, False, True, nil, 0, ''));
  // show popup
  ScreenPoint := ClientToScreen (MousePos);
  PopupMenu2.Popup (ScreenPoint.X, ScreenPoint.Y);
  Handled := True;
  // remove dynamic items
  PopupMenu1.Items [4] .Free;
  PopupMenu2.Items [3] Free;
end;
```

Ovaj primer dodaje nešto dinamičkog ponašanja iskačućem meniju dodavanjem privremenog elementa koji naznačava kada je iskačući meni prikazan. Ovo nije naročito korisno, ali sam to učinio da bih istakao način na koji možete prikazati iskačući meni kada Vam je potrebno, jer lako možete upotrebiti svojstvo PopupMenu kontrole ili njene roditeljske kontrole. Obrada događaja `OnContextMenu` ima smisla kada želite da obavite dodatnu obradu.

Parametar Handled je unapred inicijalizovan vrednošću `False`, tako da ako ništa ne uradite u obradi događaja, desiće se normalna obrada iskačućeg menija. Ukoliko nešto učinite u obradi događaja i zamenite normalnu obradu iskačućeg menija (recimo pozivanje okvira za dijalog ili menija kao u ovom slučaju), trebalo bi da parametru Handled dodelite vrednost True i sistem će prestati da obrađuje poruku. Dodeljivanje vrednosti `True` parametru `Handled` je veoma retko jer ćete češće obrađivati `OnContextPopup` da biste dinamički kreirali ili prilagodili iskačući meni, ali tada možete dopustiti da unapred određena obrada događaja zapravo prikaže meni.

Obrada događaja `OnContextPopup` nije ograničena samo na prikazivanje iskačućeg menija. Može se obaviti bilo koja operacija, recimo prikazivanje okvira za dijalog. Evo primera kako promeniti boju kontrole kada se kontrola klikne desnim tasterom miša:

```
procedure TFormPopupLabe12ContextPopup(Sender: TObject;
  MousePos: TPoint; var Handled: Boolean);
begin
  ColorDialog1.COlOr := Label2 Color;
  if ColorDialog1.Execute then
    Label2.Color := ColorDialog1.Color;
  Handled := True;
end;
```

Svi isečci koda iz ovog odeljka su dostupni u primeru CustPop.

Dinamičko kreiranje elemenata menija

Pored definisanja strukture menija pomoću Menu Designera i promenom statusa elemenata upotrebom svojstava `Checked`, `Visible` i `Caption`, možete kreirati ceo meni ili delove menija u vreme izvršavanja. Ovo ima smisla, na primer, kada imate mnogo elemenata menija koji se ponavljaju, ili kada elementi menija zavise od konfiguracije sistema ili korisničkih dozvola.

Osnovna ideja je da svaki objekat klase `TMenuItem` — koju Delphi koristi kako za elemente menija tako i za iskačuće menije — sadrži listu elemenata menija. Svaki od tih elemenata ima istu strukturu na rekurzivan način. Iskačući meniji sadrže listu podmenija, svaki podmeni listu podmenija, od kojih svaki sadrži svoju listu podmenija i tako dalje. Svojstva koja možete upotrebiti za prolazak kroz strukturu menija su `Items`, koji sadrži listu elemenata menija, i `Count`, koji sadrži broj podelemenata. Dodavanje novih elemenata menija ili menija postojećem meniju je prilično lako, naročito ukoliko možete napisati jednu rutinu za obradu događaja za sve njih.

Ovo je demonstrirano primerom DynaMenu koji takođe ilustruje upotrebu oznaka za potvrdu u meniju, opcionih elemenata i mnogih drugih karakteristika koje su detaljno opisane u tekstu. Čim pokrenete program, on kreira novi meni sa elementima koji se koriste za promenu veličine fonta velike oznake koja se nalazi na formularu. Umesto kreiranja velikog broja elemenata menija sa naslovima koji označavaju veličine između 8 i 48, možete prepustiti programu da posao obavi umesto Vas.

Novi meni bi trebalo da bude umetnut u svojstvo `Items` komponente `MainMenu`. Poziciju možete izračunati ukoliko iz komponente MainMenu zatražite prethodni meni:

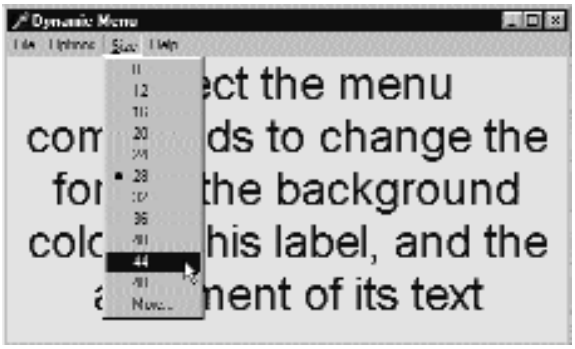
```
procedure TFormColorlText.FormCreate(Sender: TObject);
var
  PullDown, Item: TMenuItem;
  Position, I: Integer;
begin
  // create the new pull-down menu
  PullDown := TMenultem.Create (Self);
  PullDown.AutoHotkeys := maManual;
  PullDown.Caption := '&Size';
  PullDown.OnClick := SizeClick;
  // compute the position and add it
  Position MainMenu1.Items.IndexOf (Options1);
  MainMenu1.Items.Insert (Position + 1, PullDown);
  // create menu items for various sizes
  I := 8;
  while I <= 48 do
  begin
    // create the new item
    Item := TMenuItem.Create (Self);
    Item.Caption := IntToStr (I);
    // make it a radio item
    Item.GroupIndex := 1;
    Item.RadioItem := True;
    // handle click and insert
    Item.OnClick := SizeItemClick;
    PullDown.Insert (PullDown.Count, Item);
    I := I + 4;
  end;
```

```
    // add extra item at the end
    Item := TMenuItem.Create (Self);
    Item.Caption :=  'More...';
    // make it a radio item
    Item.GroupIndex := 1;
    Item.RadioItem := True;
    // handle it by showing the font selection dialog
    Item.OnClick := Font1Click;
    PullDown.Insert (PullDown.Count, Item);
  end;
```

Kao što u prethodnom kodu možete videti, elementi menija su kreirani unutar petlje `while`, u kojoj se određuje stil radio item i poziva metod `Insert` sa brojem elemenata kao parametrom za dodavanje svakog elementa na kraj menija. Na kraju program dodaje još jedan element, koji se koristi za određivanje veličine koja nije među prikazanim veličinama. Događaj `OnClick` poslednjeg elementa menija se obrađuje metodom `Font1Click` (koji je, takođe, pridružen određenom elementu menija), koji prikazuje okvir za dijalog za izbor fontova. Dinamički meni možete videti na slici 5.6.

UPOZORENJE

Pošto program dinamički koristi `Caption` novih elemenata, trebalo bi da onemogućimo svojstvo `AutoHotKeys` komponente glavnog menija ili da onemogućimo ovu funkciju za meni koji želimo da dodamo (i time ga onemogućimo za elemente menija). To je ono što sam učinio u prethodnom kodu određivanjem vrednosti `maManual` svojstva `AutoHotKeys` dinamički kreirane komponente menija. Alternativni pristup je da dopustite da meni prikaže automatske naslove i da zatim pozovete funkciju `StripHotkeys` pre nego što naslov konvertujete u broj. Takođe, postoji i nova funkcija `GetHotkeys` koja kao rezultat daje aktivan karakter naslova. ■



SLIKA 5.6 *Meni Size primera DynaMenu se kreira u vreme izvršavanja, kao i svi elementi tog menija*

Obrada događaja `OnClick` ovih dinamički kreiranih elemenata menija koristi naslov elementa menija `Sender` da bi odredila veličinu fonta:

```
procedure TFormColorText.SizeItemClick(Sender: TObject);
begin
  with Sender as TMenuItem do
    Label1.Font.Size := StrToInt (Caption);
end;
```

Ovaj kod ne određuje dobru oznaku pored odabranog elementa jer korisnik može odabrati novu veličinu i promenom veličine fonta. Odgovarajuća oznaka se prikazuje obradom događaja `OnClick` celog menija, koji je povezan odmah posle kreiranja menija, a aktiviran je odmah pre prikazivanja. Kod pretražuje elemente menija (objekat `Sender`) i proverava da li naslov odgovara trenutnoj vrednosti `Size` fonta. Ukoliko se ne pronađe odgovarajuća vrednost, program označava poslednji element menija da bi se naznačilo da je aktivna neka druga veličina:

```
procedure TFormColorText.SizeClick (Sender: TObject);
var
  I: Integer;
  Found: Boolean;
begin
  Found := False;
  with Sender as TMenuItem do
  begin
    // look for a match, skipping the last item
    for I := 0 to Count - 2 do
      if StrToInt (Items [I].Caption) =
        Label1. Font.Size then
      begin
        Items[I].Checked := True;
        Found:= True;
        System.Break; // skip the rest of the loop
      end;
    if not Found then
      Items [Count — 1].Checked := True;
  end;
end;
```

Kada želite da dinamički kreirate meni ili element menija, možete upotrebiti odgovarajuće komponente kao što sam ja to učinio u primeru DynaMenu. Kao alternativu možete upotrebiti globalne funkcije koje su na raspolaganju u jedinici Menu: `NewMenu`, `NewPopupMenu`, `NewSubMenu`, `NewItem` i `NewLine`.

Upotreba ikona u meniju

U Delphiju je veoma lako unaprediti korisnički interfejs dodavanjem slika elementima menija. Ovo postaje uobičajeno za Windows aplikacije i veoma je lepo od Borlanda što je dodao ovakvu podršku i što je razvoj grafičkih menija učinio trivijalnim.

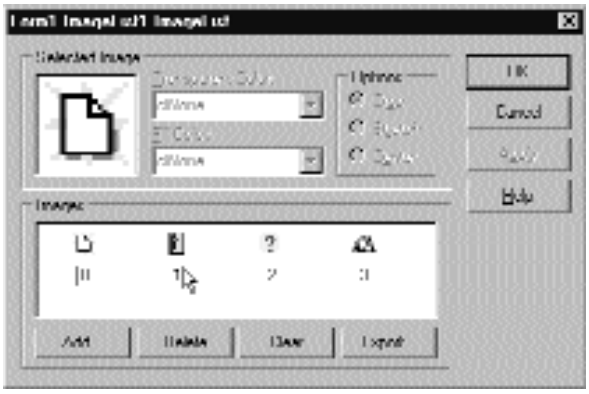
Sve što je potrebno da uradite je da dodate listu slika formularu, dodate seriju bitmapa listi slika, povežete listu slika sa menijem koristeći svojstvo menija `Images` i odredite odgovarajuće svojstvo `ImageIndex` za elemente menija. Efekat ovih jednostavnih operacija možete videti na slici 5.7. (Bitmapu možete, takođe, direktno dodeliti elementu menija koristeći svojstvo `Bitmap`.)

SAVET

Delphi 5 čini definiciju slika za menije fleksiblnijom omogućavajući Vam da dodelite listu slika bilo kom meniju (pa čak i određenom elementu menija) upotrebom novog svojstva `SubMenuImages`. Postojanje određene i manje liste slika za svaki od menija, umesto jedne velike liste slika za ceo meni, omogućava bolje prilagođavanje aplikacije u vreme izvršavanja. ■

SLIKA 5.7 *Jednostavan grafički meni primera MenuImg*

Da biste kreirali listu slika, možete dva puta kliknuti komponentu čime se aktivira odgovarajući editor (prikazan na slici 5.8), a zatim odabrati postojeće bitmape ili fajlove ikona. Možete pripremiti jednu veliku bitmapu i prepustiti editoru da je podeli prema vrednostima svojstava `Height` i `Width` komponente ImageList, koja se odnose na veličinu pojedinačnih bitmapa liste.



SLIKA 5.8 *Editor Image List sa bitmapama za primer MenuImg*

SAVET

Kao alternativu možete upotrebiti niz slika koje dobijate uz Delphi i koje se čuvaju u direktorijumu `Program Files/Common Files/ Borland Shared/Images/Buttons`. Svaka bitmapa sadrži "aktivnu" (enabled) i "neaktivnu" (disabled) verziju. Kada ih odaberete, editor Image List će pitati da li da ih podeli, što je predlog koji treba prihvatiti. Ova operacija dodeljuje listi slika normalnu sliku i neaktivnu verziju, koja se, uopšte uzev, ne koristi (ali se može automatski ugraditi po potrebi). Zbog toga sam uklonio neaktivni deo bitmape iz editora Image List. ■

Kod programa je veoma jednostavan. Jedini element koji želim da istaknem je da ukoliko podesite svojstvo `Checked` elementa menija kojem je dodeljena slika umesto prikazivanja oznake, element će biti prikazan kao utisnut. Ovo možete videti u meniju Large Font primera MenuImg prikazanog na slici 5.7. Evo koda za takav izbor elementa menija:

```
procedure TForm1.LargeFont1Click(Sender: TObject);
begin
  if Memo1.Font.Size = 8 then
    Memo1.Font.Size := 12
  else
    Memo1.Font.Size := 8;
  // changes the image style near the item
  LargeFont1.Checked := not LargeFont1.Checked;
end;
```

Prilagođavanje sistemskog menija

U nekim slučajevima je interesantno dodati komande menija samom sistemskom meniju, umesto (ili pored) postojanja linije menija. Ovo može biti korisno za sekundarne prozore, palete alata, prozore koji zahtevaju veliki prostor na ekranu i "brze i prljave" aplikacije. Dodavanje jednog menija sistemskom meniju je jednostavno:

```
AppendMenu (GetSystemMenu (Handle, FALSE),
  MF_SEPARATOR, 0, '');
AppendMenu (GetSystemMenu (Handle, FALSE),
  MF_STRING, idSysAbout, '&About...');
```

Ovaj isečak koda (izdvojen iz obrade događaja `OnCreate` primera SysMenu) dodaje separator i novi element menija elementu sistemskog menija. API funkcija `GetSystemMenu`, koja kao parametar zahteva vezu sa formularom, kao rezultat daje vezu sa sistemskim menijem. API funkcija `AppendMenu` je funkcija opšte namene koju možete upotrebiti za dodavanje elemenata menija ili celih menija bilo kom meniju (liniji menija, sistemskom meniju ili postojećem meniju). Kada dodajete element menija, potrebno je da naznačite tekst i numerički identifikator. Ja sam u primeru definisao ovaj identifikator kao:

```
const
  idSysAbout = 100;
```

Dodavanje elementa menija sistemskom meniju je lako, ali kako obraditi takav izbor? Izbor iz normalnog menija generiše Windows poruku `wm_Command`. Ona se interno obrađuje u Delphiju koji zatim aktivira događaj `OnClick` odgovarajućeg elementa menija. Izbor komande sistemskog menija, umesto toga, generiše poruku `wm_SysCommand`, koja se prosleđuje unapred određenoj rutini za obradu Delphiju. Windows obično mora da učini nešto kao odgovor na komandu sistemskog menija.

Mi možemo presresti ovu komandu i proveriti da li je identifikator komande (koji se prosleđuje poljem `CmdType` parametra `TWmSysCommand`) elementa menija naš `idSysAbout`. Pošto u Delphiju ne postoji odgovarajući događaj, potrebno je da definišemo novi metod za takvu poruku za klasu formulara:

```
public
  procedure WMSysCommand (var Msg: TMessage);
    message wm_SysCommand;
```

Kod ove procedure nije složen. Potrebno je samo da proverimo da li je komanda naša i da pozovemo unapred određenu rutinu za obradu:

```
procedure TForm1.WMSysCommand (var Msg: WMSysCommand)
begin
  if Msg.CmdType = idSysAbout then
    ShowMessage ('Mastering Delphi : SysMenu example');
  inhereted;
end;
```

Da bismo izgradili složeniji sistemski meni, umesto da dodajemo i obrađujemo pojedinačno svaki element menija kao što smo to do sada činili, možemo slediti drugačiji pristup. Dodajte samo komponentu MainMenu formularu, kreirajte strukturu komponente (bilo kakva struktura će poslužiti) i napišite odgovarajuće rutine za obradu događaja. Zatim resetujte vrednost svojstva `Menu` formulara uklanjajući liniju menija.

Sada možemo dodati kod primeru SysMenu kojim se dodaje svaki element sakrivenog menija sistemskom meniju. Ova operacija se izvršava kada se klikne kontrola formulara. Odgovarajuća obrada koristi generički kod koji ne zavisi od strukture menija koji dodajemo sistemskom meniju:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  // add a separator
  AppendMenu (GetSystemMenu (Handle, FALSE), MF_SEPARATOR, 0, '');
  // add- the-main menu to thesystenrweno
  with MainMenu1 do
    for I := 0 to Items.Count - 1 do
       AppendMenu (GetSystemMenu (Self.Handle, FALSE),
         mf_Popup, Items[I].Handle, PChar (Items[I].Caption));
  // disable the button
  Button1.Enabled := False;
end;
```

SAVET

Ovaj kod koristi izraz `Self.Handle` da bi pristupio hendlu formulara. Ovo je neophodno jer mi trenutno radimo sa komponentom `MainMenu1`, što je naznačeno iskazom `with`. ■

Zastavica menija koja se koristi u ovom slučaju, zastavica `mf_Popup`, označava da dodajemo meni. U pozivu ove funkcije četvrti parametar se interpretira kao hendl menija koji dodajemo (u prethodnom primeru smo umesto toga prosledili identifikator menija). Pošto sistemskom meniju dodajemo elemente menija koji sadrže podmenije, konačna struktura sistemskog menija će imati dva nivoa, što se može videti na slici 5.9.

SLIKA 5.9 *Drugi nivo elemenata sistemskog menija primera SysMenu je rezultat kopiranja kompletnog glavnog menija u sistemski meni*

UPOZORENJE

Windows API koristi termine pop-up meni i pull-down meni naizmenično. Ovo je veoma čudno jer većina od nas koristi termine koji imaju različito značenje. Pop-up meniji su iskačući meniji, a pull-down meniji su sekundarni meniji linije menija. Izgleda da Microsoft koristi termine na ovaj način jer su dva elementa implementirana uz istu vrstu internih prozora, a činjenica da predstavljaju dva različita korisnička interfejsa je verovatno nešto što je kasnije konceptualno izgrađeno nad jednom osnovnom internom strukturom. ■

Kada ste dodali elemente menija sistemskom meniju, potrebno je da ih obradite. Naravno, svaki element menija možete proveriti metodom `WMSysCommand`, ili možete pokušati da načinite pametniji pristup. Pošto je u Delphiju lakše napisati rutinu za obradu događaja `OnClick` za svaki element, možemo potražiti odgovarajući element za dati identifikator u strukturi menija. Delphi nam pomaže jer obezbeđuje metod `FindItem`.

Kada (i ukoliko) pronađemo element glavnog menija koji odgovara odabranom elementu sistemskog menija, možemo pozvati odgovarajući metod `Click` (koji poziva obradu `OnClick`). Evo koda koji sam dodao metodu `WMSysCommand`:

```
var
  Item: TMenuItem;
begin
  ...
  Item := MainMenu1.FindItem (Msg.CmdType, fkCommand);
  if Item <> nil then
    Item.Click;
```

U ovom kodu polje `CmdType` strukture poruke se prosleđuje proceduri `WMSysCommand` koja sadrži komandu elementa menija koji se poziva.

NAPOMENA

Takođe, možete upotrebiti jednostavan iskaz `if` ili `case` da biste obradili jedan od unapred određenih elemenata sistemskog menija koji kao identifikatore imaju specijalne kodove, recimo, `sc_Close`, `sc_Minimize`, `sc_Maximize` i tako dalje. Za više informacija možete pogledati opis poruke `wm_SysCommand` u Windows API help fajlu. ■

Ova aplikacija funkcioniše, ali postoji jedan propust. Ukoliko desnim tasterom miša kliknete ikonu koja na Taskbaru predstavlja aplikaciju, prikazaće se samo sistemski meni (zapravo drugačiji od unapred definisanog). Razlog tome je što ovaj sistemski meni pripada drugom prozoru, prozoru globalnog objekta `Application`. Razmatraću objekat `Application` u Poglavlju 6 i tada ću ažurirati ovaj primer tako da funkcioniše i kontrola na Taskbaru.

Komponenta ActionList

Kao što sam objasnio u prethodnom poglavlju, Delphijeva arhitektura događaja je veoma otvorena. Možete napisati jednu obradu događaja i pridružiti je događajima `OnClick` kontrole palete alata i menija. Takođe, istu obradu događaja možete pridružiti različitim kontrolama ili elementima menija, jer obrada događaja može koristiti parametar `Sender` da se referišete na objekat koji je pokrenuo događaj koristeći parametar `Sender`. Nešto je teže sinhronizovati status kontrola palete alata i elemenata menija. Ukoliko imate element menija i kontrolu palete alata koji se odnose na istu opciju, svaki put kada opcija promeni stauts, morate dodati i oznaku elementu menija i promeniti status kontrole da biste je prikazali utisnutom.

Da bi se prevazišao ovaj problem, Delphi 4 je predstavio arhitekturu obrade događaja zasnovanu na akcijama. Akcija (ili komanda) označava operaciju koju treba obaviti kada se klikne element menija ili kontrola i određuje status svih elemenata koji su u vezi sa akcijom. Veza akcije sa korisničkim interfejsom povezane kontrole je veoma važna i ne treba je potcenjivati jer je to mesto gde možete iskoristiti prednosti ovakve arhitekture.

NAPOMENA

Ukoliko ste ikada napisali kod koristeći MFC biblioteku Visual C++, prepoznaćete da Delphi akcija mapira i komandu i objekat `CCommandUpdateUI`. Delphi arhitektura je mnogo fleksibilnija jer se može proširiti formiranjem potklasa klasa akcija. ■

Postoji mnogo učesnika u ovoj arhitekturi obrade događaja. Centralna uloga svakako pripada objektima akcija. Objekti akcija imaju naziv, kao i bilo koja druga komponenta, i poseduju svojstva koja će biti primenjena na povezane kontrole (koje se nazivaju klijentima akcije). Ova svojstva uključuju `Caption`, grafičku reprezentaciju (`ImageIndex`), status (`Checked`, `Enabled` i `Visible`) i korisničku informaciju (`Hint` i `HelpContext`). Osnovna klasa objekta akcije je klasa `TBasicAcion`. Postoji i klasa `TAction`, ali je ona izvedena iz klase `TCustomAction`, koja je izvedena iz klase `TContainedAction`, koja je izvedena iz `TBasicAction`, potklase `TComponent`.

Svaki objekat akcije je povezan sa jednim ili više klijent objekata preko objekta ActionLink. Više kontrola, moguće i različitog tipa, može deliti isti objekat akcije što je naznačeno svojstvom `Action`. Objekti ActionLink održavaju bidirekcionu vezu između klijent objekta i akcije. Objekat ActionLink je neophodan jer veza funkcioniše u oba smera. Operacija nad objektom (recimo kada kliknete) se prosleđuje objektu akcije i dovodi do poziva događaja `OnExecute`; ažuriranje statusa objekta akcije se reflektuje na povezane klijent objekte. Drugim rečima, jedna ili više kontrola može kreirati ActionLink, koji se registruje za objekat akcije.

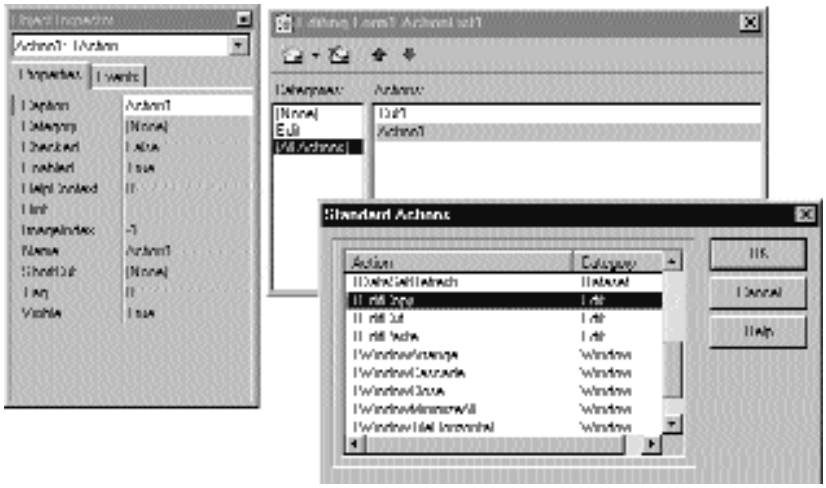
Ne bi trebalo da određujete svojstva klijent kontrola koje povezujete sa akcijom jer će akcija zaobići vrednosti svojstava klijent kontrola. Zbog toga bi prvo trebalo da napišete akcije, a da zatim kreirate elemente menija i kontrole koje želite da povežete. Takođe, imajte na umu da kada

akcija nema obradu `OnExecute`, klijent kontrola je automatski neaktivna (ili prikazana sivom bojom), izuzev ukoliko nije dodeljena vrednost `False` svojstvu `DisableIfNoHandler`.

Klijent kontrole povezane sa akcijom su obično elementi menija i različiti tipovi kontrola (push kontrole, polja za potvrdu, opcione kontrole, kontrole palete alata i slične kontrole), ali Vas ništa ne sprečava da kreirate nove komponente koje se mogu umetnuti u ovu strukturu. Oni koji pišu komponente mogu čak definisati nove akcije i nove objekte akcije veze.

Pored klijent kontrola, neke akcije takođe mogu imati ciljne komponente. Neke unapred određene akcije su povezane sa određenom ciljnom komponentom (na primer, pogledajte opis komponenata DataSet u odeljku "Pronalaženje slogova u tabeli" u Poglavlju 9). Druge akcije automatski pretražuju komponente formulara koje podržavaju datu akciju, počevši od aktivne kontrole.

Konačno, objekti akcija se čuvaju u komponenti ActionList, jedinoj klasi ove arhitekture koja se prikazuje na Components Palette. Lista akcija sadrži izvršne akcije koje nisu obrađene određenim objektom akcije, pozivajući `OnExecuteAction`. Iako lista akcija ne obrađuje akciju, Delphi poziva događaj `OnExecuteAction` objekta `Application`. Komponenta ActionList ima specijalni editor koji možete koristiti da biste kreirali akcije, kao što možete videti na slici 5.10.



SLIKA 5.10 *Editor komponente ActionList, koji prikazuje unapred određene akcije koje možete koristiti*

U editoru se akcije prikazuju u zasebnim grupama kao što je naznačeno svojstvom `Category`. Jednostavnim dodeljivanjem potpuno nove vrednosti ovom svojstvu Vi nalažete editoru da uvede novu kategoriju. Ove kategorije su u osnovi logičke grupe, mada se u nekim slučajevima grupa akcija može obavljati nad određenim tipom ciljne komponente. Možda želite da definišete kategoriju za svaki meni ili da ih grupišete na neki drugi logički način.

Upotrebom editora liste akcija možete kreirati potpuno novu akciju ili odabrati jednu od postojećih akcija koje su registrovane na sistemu. Ove akcije su prikazane u sekundarnom okviru za dijalog prikazanom na slici 5.10. Postoji veliki broj unapred određenih akcija koje mogu da budu podeljene u logičke grupe:

- **AKCIJE EDITOVANJA** (Edit actions), ilustrovane narednim primerom. Ove akcije uključuju Cut, Copy i Paste.
- **AKCIJE MDI PROZORA** (MDI window actions), koje će biti demonstrirane u Poglavlju 8 kada budemo razmatrali pristup Multiple Document Interface. Sadrže sve najuobičajenije MDI operacije: Arrange, Cascade, Close, Tile i Minimize all.
- **AKCIJE SKUPA PODATAKA** (Dataset actions), koje se odnose na tabele baze podataka i upite, a biće razmatrane u Poglavlju 11. Postoje mnoge akcije skupa podataka koje predstavljaju sve glavne operacije koje možete obaviti nad skupom podataka.
- **HELP AKCIJE** (Help actions), koje Vam omogućavaju da aktivirate stranu sadržaja ili indeks Help fajla koji je pridružen aplikaciji.

NAPOMENA

Takođe, možete definisati sopstvene akcije i registrovati ih u Delphi IDE-u što ćemo videti u Poglavlju 13. ■

Pored obrade događaja `OnExecute` akcije i promene statusa akcije da bi se uticalo na korisnički interfejs klijent kontrola, akcija takođe može obraditi događaj `OnUpdate` koji se aktivira kada aplikacija nije aktivna. To Vam daje mogućnost da proverite status aplikacije ili sistema i da promenite kontrole korisničkog interfejsa po potrebi. Na primer, standardna akcija PasteEdit aktivira klijent kontrole samo kada postoji tekst na Clipboardu.

Akcije u praksi

Sada kada razumete osnovne ideje koje stoje iza ove veoma važne Delphi karakteristike, isprobajmo primer. Program je nazvan Actions i demonstrira veliki broj karakteristika arhitekture akcija.

Počeo sam sa izradom postavljanjem nove komponente ActionList na formular primera, dodavanjem tri standardne akcije editovanja i dodavanjem nekoliko sopstvenih akcija. Ovaj formular sadrži i panel sa nekoliko točkića, glavnim menijem i Memo kontrolom (automatskim ciljem edit akcija). Ovo je lista akcija koja je izdvojena iz DFM fajla:

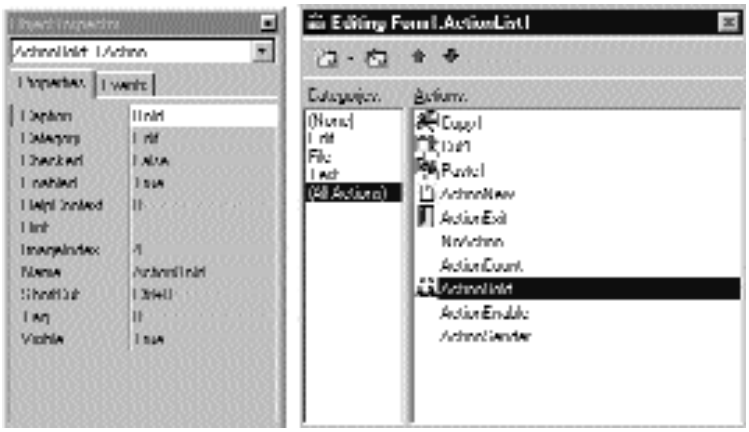
```
Object ActionList1: TActionList
  Images = ImageList1
  object ActionCopy: TEditCopy
    Category = 'Edit'
    Caption = '&Copy'
    Hint = 'Copy'
    ImageIndex = 1
    ShortCut = <Ctrl+C>
  end
  object ActionCut: TEditCut
    Category = 'Edit'
    Caption = 'Cu&t'
    Hint = 'Cut'
    ImageIndex = 0
    ShortCut = <Ctrl+X>
  end
  object ActionPaste: TEditPaste
    Category = 'Edit'
```

```
    Caption = '&Paste'
    Hint = 'Paste'
    ImageIndex = 2
    ShortCut = <Ctrl+V>
  end
  object ActionNew: TAction
    Category = 'File'
    Caption = '&New'
    Hint = 'New'
    ImageIndex = 3
    ShortCut = <Ctrl÷N>
    OnExecute = ActionNewExecute
  end
  object ActionExit: TAction
    Category = 'File'
    Caption = 'E&xit'
    Hint = 'Exit'
    ImageIndex = 5
    ShortCut = <Alt+F4>
    OnExecute = ActionExitExecute
  end
  object NoAction: TAction
    Category = 'Test'
    Caption = '&No Action'
    Hint = 'No Action'
  end
  object ActionCount: TAction
    Category = 'Test'
    Caption = '&Count Chars'
    Hint = 'Count Characters'
    ImageIndex = 6
    OnExecute = ActionCountExecute
    OnUpdate = ActionCountUpdate
  end
  object ActionBold: TAction
    Category = 'Edit'
    Caption = '&Bold'
    Hint = 'Bold'
    ImageIndex = 4
    ShortCut = <Ctrl+B>
    OnExecute = ActionBoldExecute
  end
  object ActionEnable: TAction
    Category = 'Test'
    Caption = '&Enable NoAction'
    Hint = 'Enable No Action'
    OnExecute = ActionEnableExecute
  end
  object ActionSender: TAction
    Category = 'Test'
    Caption = 'Test &Sender'
    Hint = Test Sender
    OnExecute = ActionSenderExecute
  end
end
```

NAPOMENA

Tastaturne prečice se čuvaju u DFM fajlu kao virtuelni brojevi koji, takođe, naznačavaju vrednosti za tastere Ctrl i Alt. U ovom i ostalim listinzima ove knjige ja sam brojeve zamenio literalima, ograničavajući ih uglastim zagradama. ■

Sve ove akcije su povezane sa elementima komponente MainMenu, a neke od njih su povezane sa kontrolama kontrole Toolbar (više o kontroli Toolbar u Poglavlju 7). Primetite da slike selektovane u kontroli ActionList utiču samo na akcije u editoru, kao što možete videti na slici 5.11. Da bi se prikazale i slike iz ImageLista za elemente menija i kontrole palete alata, morate, takođe, selektovati listu slika u komponentama MainMenu i Toolbar.



SLIKA 5.11 *Editor ActionList primera Actions*

Tri unapred određene akcije menija Edit nemaju pridruženu obradu, ali ovi specijalni objekti imaju interni kod za izvršavanje odgovarajućih akcija kontrole aktivnog editovanja ili kontrole Memo. Ove akcije, takođe, aktiviraju ili deaktiviraju same sebe u zavisnosti od sadržaja Clipboarda i prema postojanju selektovanog teksta aktivne kontrole. Većina ostalih akcija sadrži korisnički kod, izuzev objekta `NoAction`. Kako ne postoji kod, element menija i kontrola koji su povezani sa ovom komandom su neaktivni, iako je svojstvu `Enabled` dodeljena vrednost True.

Ja sam primeru, i meniju Test, dodao još jednu akciju koja aktivira element menija povezan sa objektom `NoAction`:

```
procedure TForm1.ActionEnableExecute (Sender: TObject);
begin
  NoAction.Enabled := True;
  NoAction.DisableIfNoHandler := False;
  ActionEnable := False;
end;
```

Dodeljivanje vrednosti `True` svojstvu `Enabled` će proizvesti efekat za veoma kratko vreme, izuzev ukoliko ne podesite svojstvo `DisableIfNoHandler`, kao što smo razmatrali u prethodnom odeljku. Kada se operacija izvršila, ja sam onemogućio aktuelnu akciju jer nema potrebe za ponovnim zahtevom za istom komandom.

Ovo je različito od akcije koju možete uključiti i isključiti, kao što je element menija Edit➥Bold i odgovarajući točkić. Evo koda za akciju Bold:

```
procedure TForm1.ActionBoldExecute(Sender: TObject);
begin
  with Memo1.Font do
    if fsBold in Style then
      Style := Style - [fsBold]
    else
      Style := Style + [fsBold];
  // toggle status
  ActionBold.Checked := not ActionBold.Checked
end;
```

Objekat `ActionCount` sadrži veoma jednostavan kod, ali demonstrira obradu `OnUpdate`; kada je Memo kontrola prazna, ona je automatski neaktivna. Isti efekat bismo mogli da proizvedemo obradom događaja `OnChange` same Memo kontrole, ali, uopšte uzev, ne mora biti moguće ili se ne može lako odrediti status kontrole jednostavnom obradom jednog od događaja. Evo koda obe obrade akcije:

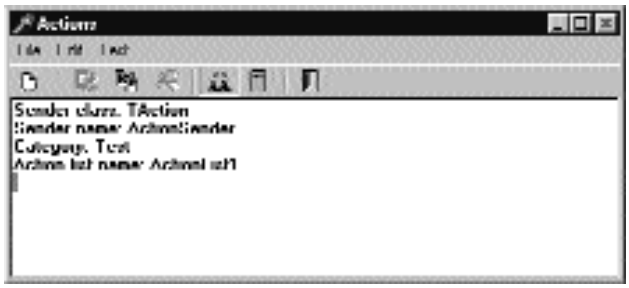
```
procedure TForm1.ActionCountExecute(Sender TObject);
begin
  ShowMessage ( 'Characters:' + IntToStr (
    Length (Memo1.Text));
end;

procedure TForm1.ActionCountUpdate(Sender: TObject);
begin
  ActionCount.Enabled := Memol.Text <> '';
end;
```

Konačno, dodao sam specijalnu akciju kojom se testira objekat Sender obrade događaja akcije čime sam dobio sistemske informacije. Pored prikazivanja naziva i klase objekta, ja sam dodao kod kojim se pristupa objektu liste akcija. To sam uglavnom učinio da bih prikazao da možete pristupiti ovoj informaciji i kako da to učinite:

```
procedure TForm1.ActionSenderExecute(Sender: TObject);
begin
  Memo1.Lines.Add (
    'Sender class:' + Sender.ClassName)
Memo1.Lines.Add (
    'Sender name:' + (Sender as TComponent).Name);
Memo1.Lines.Add (
    'Category:' + (Sender as TAction).Category);
Memo1.Lines.Add (
    'Action list name:' + (Sender as TAction).ActionList.Name);
end;
```

Izlaz ovog koda možete videti na slici 5.12 kao i korisnički interfejs ovog primera. Primetite da `Sender` nije element menija koji ste selektovali, iako mu je pridružena obrada događaja. Objekat `Sender`, koji inicira događaj, je akcija koja presreće korisničku akciju.

Konačno, imajte na umu da možete napisati i obrade događaja za sam objekat ActionList koji igra ulogu globalnih obrada za sve akcije liste (nešto što nisam uradio u prethodnom primeru).



SLIKA 5.12 *Primer Actions sa detaljnim opisom Sendera događaja OnExecute objekta Action*

Kontrole koje iscrtava vlasnik

Vratimo se nakratko na grafiku u menijima. Pored upotrebe ImageLista za elemente menija, meni možete pretvoriti u potpuno grafički element upotrebom tehnike kada vlasnik iscrtava kljenta. Ista tehnika funkcioniše i za ostale kontrole, recimo, za liste. U Windowsu je obično sistem odgovoran za iscrtavanje kontrola, lista, polja za izmene, elemenata menija i sličnih elemenata. U osnovi, ove kontrole znaju kako da same sebe iscrtaju. Kao alternativu, ipak, sistem omogućava vlasniku ovih kontrola, uopšte uzev formularu, da ih iscrta. Ova tehnika, koja je moguća za kontrole, liste, combo polja i elemente menija, se naziva *owner-draw*.

U Delphiju je situacija nešto složenija. Komponente se mogu postarati o sopstvenom iscrtavanju u ovom slučaju (kao kod klase `TBitBtn` za bitmapirane kontrole) i moguće je da aktiviraju odgovarajuće događaje. U osnovi sistem šalje zahtev za iscrtavanjem vlasniku (obično formularu), a formular prosleđuje događaj odgovarajućoj kontroli, inicirajući njenu obradu događaja.

NAPOMENA

Većina Win32 kontrola ima podršku za tehniku owner-draw, koja se uobičajeno naziva iscrtavanjem. U potpunosti možete prilagoditi ListView, TreeView, TabControl, PageControl, HeaderControl, StatusBar i ToolBar. U Delphiju 5 kontrole ToolBar, ListView i TreeView takođe podržavaju napredno iscrtavanje, fino podešenu mogućnost iscrtavanja koju je predstavio Microsoft u poslednjim verzijama Win32 biblioteka kontrola. Nedostatak iscrtavanja je u tome da kada se stil korisničkog interfejsa Windowsa promeni u budućnosti (a to se uvek dešava), takve kontrole — koje se perfektno uklapaju u trenutni korisnički stil — neće izgledati dobro. Kako kreirate korisnički interfejs, potrebno je da ga stalno ažurirate. Nasuprot tome, ukoliko koristite standardni izlaz kontrola, Vaše aplikacije će se automatski adaptirati novim verzijama kontrola. ■

Elementi menija koje iscrtava vlasnik

Delphi čini razvoj grafičkih elemenata menija prilično jednostavnim u poređenju sa tradicionalnim pristupom Windows API-ja. Vi ćete dodeliti vrednost `True` svojstvu `OwnerDraw` elementa menija i obradićete događaje `OnMeasureItem` i `OnDrawItem`.

U događaju `OnMeasureItem` možete odrediti veličinu elemenata menija. Ova obrada događaja se aktivira jednom za svaki element menija kada se prikaže meni, a sadrži dva parametra reference koja možete podesiti:

```
procedure ColorMeasureItem (Sender: TObject;
  ACanvas: TCanvas; var Width, Height: Integer);
```

Drugi parametar, parametar `ACanvas`, se tipično koristi za određivanje veličine aktuelnog fonta.

U događaju `OnDrawItem` možete iscrtati sliku. Ova obrada događaja se aktivira svaki put kada element treba ponovo iscrtati. To se dešava kada Windows prvi put prikaže elemente i svaki put kada se status promeni; na primer, kada se mišem pređe preko elementa, element treba istaknuti. Zapravo, da bismo iscrtali element menija, moramo uzeti u obzir sve mogućnosti, uključujući iscrtavanje istaknutih elemenata određnom bojom, iscrtavanje oznake za potvrdu ukoliko je potrebno i tako dalje. Srećom, Delphi događaj prosleđuje obradi `Canvas` gde je potrebno izvršiti iscrtavanje, izlazni četvorougao i status elementa (bilo da je selektovan ili ne):

```
procedure ColorDrawItem (Sender: TObject;
  ACanvas: TCanvas; ARect: TRect; Selected: Boolean);
```

U primeru ODMenu ja sam obradio boju isticanja, ali sam preskočio sve druge napredne aspekte (kao što su oznake za potvrdu). Podesio sam svojstvo menija `OwnerDraw` i napisao sam obrade za neke elemente menija. Da bih napisao jednu obradu za svaki događaj od tri obojena elementa, ja sam za vrednost svojstva `Tag` odredio aktuelnu boju u obradi `OnCreate` događaja formulara:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Blue1.Tag := clBlue;
  Red1.Tag := clRed;
  Green1.Tag := clGreen;
end;
```

Ovo čini obradu aktuelnog `OnClick` događaja elemenata veoma jednostavnom:

```
procedure TForm1.ColorClick(Sender: TObject);
begin
  ShapeDemo.Brush.Color :=
    (Sender as TComponent).Tag
end;
```

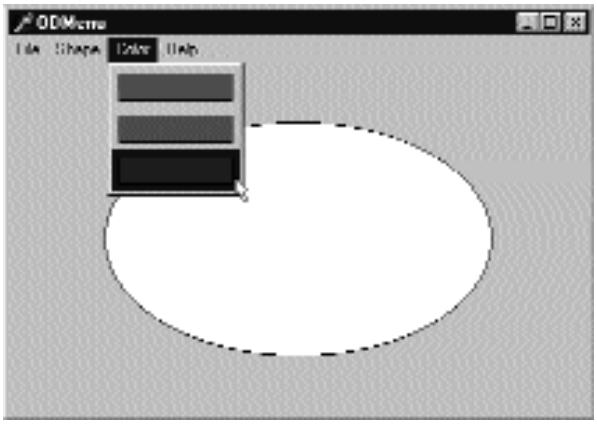
Obrada događaja `OnMeasureItem` ne zavisi od aktuelnih elemenata, ali koristi fiksnu vrednost (različitu od obrade drugih menija):

```
procedure TForm1.ColorMeasureItem(Sender: TObject;
  ACanvas: TCanvas; var Width, Height: Integer);
begin
  Width := 80;
  Heiqht := 30;
end;
```

Najvažniji deo koda su obrade događaja `OnDrawItem`. Za boju koristimo vrednost taga da bismo iscrtali četvorougao date boje, kao što možete videti na slici 5.13. Ipak, pre nego što to učinimo, potrebno je da ispunimo pozadinu elemenata menija (četvorougaona oblast koja se prosleđuje kao parametar) standardnom bojom menija (`clMenu`) ili selektovanih elemenata menija (`clHighlight`):

```
procedure TForm1.ColorDrawItem(Sender: TObject;
  ACanvas: TCanvas; ARect: TRect; Selected: Boolean);
begin
```

```
  // set the background color and draw it
  if Selected then
    ACanvas.BruSh.Color := clHighllght
  else
    ACanvas.Brush.Color := clMenu;
  ACanvas. FillRect (ARect);
  // show the color
  ACanvas.Brush.Color := (Sender as TComponent).Tag;
  Inflate.Rectangle (ARect, -5, -5);
  ACanvas.Rectangle (ARect.Left, ARect.Top,
    ARect.Right, ARect.Bottom);
end;
```



SLIKA 5.13 *Meni koji iscrtava vlsnik primera ODMenu*

Tri obrade za ovaj događaj elemenata menija Shape su različite, mada koriste sličan kod:

```
procedure TForm1.Ellipse1DrawItem(Sender: TObject;
  ACanvas: TCanvas; ARect: TRect; Selected: Boolean);
begin
  // set the background color and draw it
  if Selected then
    ACanvas.Brush.Color := clHighlight
  else
    ACanvas.Brush.Color := clMenu;
  ACanvas.FillRect (ARect);
  // draw the ellipse
  ACanvas.Brush.Color := clWhite;
  InflateRect (ARect, -5, -5);
  ACanvas.Ellipse (ARect.Left, ARect.Top, ARect.Right, ARect.Bottom);
end;
```

NAPOMENA

Da bi mogao da prihvati sve veći broj stanja korisničkog interfejsa Windowsa 2000, Delphi 5 sadrži novi događaj za menije, `OnAdvancedDrawItem`. ■

Raznobojni ListBox

Kao što smo videli kod menija, ListBoxovi imaju mogućnost da ih iscrtava vlasnik, što znači da program može iscrtati elemente liste. Ista vrsta podrške postoji i za combo polja. Da bismo kreirali ListBox koji iscrtava vlasnik, za svojstvo `Style` ćemo odrediti `lbOwnerDrawFixed` ili `lbOwnerDrawVariable`. Prva vrednost označava da ćemo visinu elemenata liste odrediti dodeljivanjem vrednosti svojstvu ItemHeight i da će to biti visina za svaki od elemenata. Drugi stil iscrtavanja označava listu sa elementima različite visine. U ovom slučaju komponenta će inicirati događaj `OnMeasureItem` za svaki element, da bi se od programa dobila njihova visina.

U primeru ODList ja sam zadržao prvi, jednostavniji pristup. Primer čuva informacije o boji uz elemente liste, a zatim iscrtava elemente u zadatoj boji (umesto da se koristi jedna boja za celu listu). Evo svojstava komponenata glavnog formulara ovog primera:

```
object ODListForm: TODListForm
  Caption = 'Owner draw Listbox'
  OnCreate = FormCreate
  object ListBoil: TListBox
    Align = alClient
    Font.Charset = ANSI_CHARSET
    Font.Color = clBlack
    Font.Height = -32
    Font.Name = 'Arial'
    Font.Style [fsBold]
    Item.Height = 16
    ParentFont = False
    Sorted = True
    Style = lbOwnerDrawFixed
    OnDblClick = ListBoxlDblClick
    OnDrawltem = ListBoxlDrawltem
  end
  object ColorDialog1: TColorDialog...
end
```

Obratite pažnju na vrednost atributa formulara `TextHeight`, koji označava broj piksela koji je potreban za prikazivanje teksta. To je vrednost koju bi trebalo da upotrebimo za svojstvo ItemHeight liste. Alternativno rešenje je da ovu vrednost izračunamo u vreme izvršavanja, tako da ukoliko kasnije promenimo veličinu fonta, ne moramo da podesimo i visinu elemenata.

NAPOMENA

Upravo sam opisao `TextHeight` kao atribut formulara, ne kao svojstvo, a zapravo to i nije svojstvo već lokalna vrednost formulara. Ukoliko nije svojstvo, možete se s pravom pitati kako ga Delphi čuva u `DFM` fajlu? Dakle, odgovor leži u tome da je Delphi streaming mehanizam zasnovan na svojstvima uz dodatne klonove svojstava koje kreira metod `DefineProperties`. Možete pogledati Delphi Help fajl ili "Delphi priručnik za programere" za informacije o ovoj naprednoj temi. ■

Pošto `TextHeight` *nije* svojstvo, mada je prikazano sa opisom formulara, ne možemo mu direktno pristupiti. Proučavajući izvorni kod VCL, ja sam pronašao da se ova vrednost izračunava pozivom privatnog metoda formulara `GetTextHeight`. Kako je privatna, mi ovu funkciju ne možemo pozvati. Ono što možemo učiniti je da dupliramo kod (a on je prilično jednostavan) u metod `FormCreate` formulara, posle selektovanja fonta liste:

```
Canvas.Font := ListBox1.Font;
ListBox1.ItemHeight := Canvas.TextHeight ('0');
```

Sledeća stvar koju je potrebno da učinimo je da dodamo fontove u listu. Pošto je ovo lista boja, želimo da dodamo nazive boja za `Items` liste i odgovarajuće vrednosti boja u `Objects` koji odgovaraju svakom objektu liste. Umesto da dve vrednosti dodajem odvojeno, ja sam napisao proceduru za dodavanje novog elementa listi:

```
procedure TODListForm.AddColors (Colors: array of TColor);
var
  I: Integer;
begin
  for I := Low (Colors) to High (Colors) do
    ListBox1.Items.AddObject (
      ColorToString (Colors[I]),
      TObject(Colors[I]));
end;
```

Ovaj metod koristi prametar otvorenog niza, niz neodređenog broja elemenata istog tipa. (Pogledajte uputstvo na mreži *Essential Pascal* na adresi `www.marcocantu.com` ukoliko niste upoznati sa strukturom jezika.) Za svaki element koji se prosleđuje kao parametar, mi dodajemo naziv boje listi i dodajemo njenu vrednost odgovarajućim podacima pozivajući metod `AddObject`. Da bismo dobili string koji odgovara boji, pozivamo funkciju `ColorToString`. Ovim se dobija string koji ili sadrži odgovarajuću konstantu boje, ukoliko postoji, ili heksadecimalnu vrednost boje. Podaci o boji se dodaju listi posle konverzije njene vrednosti `TObject` tipu podataka (četvorobajtnoj referenci), kao što zahteva metod `AddObject`.

SAVET

Pored funkcije `ColorToString`, koja konvertuje vrednost u odgovarajući string sa identifikatorom ili heksadecimalnom vrednošću, postoji i Delphi funkcija za konvertovanje pravilno formatiranog stringa u boju, funkcija `StringToColor`. ■

U primeru ODList, metod se poziva iz obrade događaja `OnCreate` formulara (pošto se prethodno odredi veličina elemenata):

```
AddColors ([clRed, clBlue, clYellow, clGreen, clFuchsia, clLime,
  clGray, RGB (213, 23, 123), RGB (0, 0, 0),
  clAqua, clNavy, cl0live, clPurple, clTeal]);
```

Kod koji se koristi za iscrtavanje elemenata nije naročito složen. Jednostavno ćemo preuzeti boju koja je dodeljena elementu, odrediti je za boju fonta i zatim iscrtati tekst:

```
procedure TODListForm.ListBox1DrawItem(Control TWinControl;
  Index: Integer; Rect: TRect; State: TOwnerDrawState);
begin
  with Control as TListbox do
  begin
    // erase
    Canvas.FillRect(Rect);
    // draw item
    Canvas.Font.Color := TColor (Items.Objects [Index]);
    Canvas.TextOut(Rect.Left, Rect.Top, Listbox1.Items[Index]);
  end;
end;
```

Sistem će već odrediti odgovarajuću boju pozadine, tako da se selektovani element pravilno prikaže a da mi ne moramo dodati nikakav kod. Primer izlaza ovog programa na početku izvršavanja možete videti na slici 5.14. Primer Vam, takođe, omogućava dodavanje novih elemenata tako što ćete dva puta kliknuti listu:

```
procedure TODListForm.ListBox1DblClick(Sender: Tobject);
begin
  if ColorDialog1.Execute then
    AddColors ([ColorDialog1.Color]);
end;
```

Ukoliko pokušate da upotrebite ovu mogućnost, primetićete da se neke boje koje dodajete pretvaraju u nazive boja (jedna od Delphi konstanti boja) dok se ostale pretvaraju u heksadecimalne brojeve.

SLIKA 5.14 *Izlaz primera ODList, sa obojenom listom koju iscrtava vlasnik*

ListView i TreeView

Mada je upotreba liste koju iscrtava vlasnik prilično jednostavna, ova vrsta liste se često zamenjuje moćnijim kontrolama ListView i TreeView. Ponovimo, ove dve kontrole su deo Win32 kontrola, a čuvaju se u biblioteci ComCtl32.DLL.

Microsoft je nastavio da proširuje ovu biblioteku u protekle dve godine, dodajući nove kontrole kao što su kalendar i Coolbar, koje su dostupne još od Delphija 4, i proširujući postojeće. Neke verzije biblioteke (koje se distribuiraju uz određene brojne verzije Microsoft Internet Explorera) donele su probleme kompatibilnosti kontrola, mada je situacija izgleda postala stabilnija protekle godine.

Neke od ovih kontrola su složene, mogu se prilagoditi na brojne načine, pa čak mogu podržati i iscrtavanje. Ovde ću Vam prikazati nekoliko jednostavnih primera upotrebe komponenata TreeView i ListView. U poglavljima 7 i 8 koristićemo i druge kontrole. U svakom slučaju, ja ne mogu pružiti obiman opis svih karakteristika ovih kontrola, jer to zahteva mnogo prostora.

Grafička referentna lista

Kada koristite komponentu ListView, Vi obezbeđujete bitmape koje naznačavaju status elementa (na primer, selektovanog elementa) i opisujete sadržaj elementa na grafički način.

Kako da povežemo slike u listu ili drvo? Potrebno je da se referišemo na komponentu ImageList koju smo već koristili za slike menija. ListView može, zapravo, sadržati tri liste slika, jednu za velike ikone (svojstvo `LargeImages`), jednu za male ikone (svojstvo `SmallImages`) i jednu koja se koristi za stanje elemenata (svojstvo `StateImages`).

Da bih definisao slike primera RefList, ja sam koristio drugačiji pristup: kreirao sam jednu bitmapu (16 x 80 piksela za pet malih slika i 32 x 160 piksela za pet velikih slika) koja sadrži sve slike. Na slici 5.15 su prikazane ove dve bitmape u Delphi Image Editoru. Zatim sam dodao bitmapu resurs fajlu i napisao sam nešto koda da bih ih sve odjednom učitao (ne jednu po jednu).

SLIKA 5.15 *Sve slike za ListView primera RefList se čuvaju u dve bitmape*

Ja sam kreirao dve komponente ImageList u vreme izvršavanja. Kao što možete videti iz parametra konstruktora `Create`, ja sam kao njihovog vlasnika označio formular, tako da ne moram na kraju ručno da ih uklanjam. Evo koda obrade za prvi deo događaja formulara `OnCreate`:

```
procedure TForm1.FormCreate (Sender: TObject);
var
  ImageList1, ImageList2: TImageList;
begin
  // load the small images
  ImageList1 := TImageList.Create (self);
  ImageList1.Height := 32;
  ImaqeList1.Width := 32;
  ImageList1.ResourceLoad (rtBitmap,
```

```
    'LargeImages', clWhite);
  ListView1.LargeImages := ImageList1;

  // load the small images
  ImageList2 := TImageList.Create (self);
  ImageList2 := ResourceLoad (rtBitmap,
    'SmallImages', clWhite);
  Listview1.SmallImaqes := ImageList2;
```

Svaki od elemenata ListView ima `ImageIndex`, koji se odnosi na njegovu sliku u listi. Da bi ovo pravilno funkcionisalo, elementi dve liste slika bi trebalo da slede isti redosled. Kada popravite listu slika, možete joj dodati elemente koristeći Delphijev ListView Item Editor, koji je povezan sa svojstvom `Items`. Primer upotrebe ovog editora možete naći na slici 5.16. U ovom editoru možete definisati elemente i takozvane podelemente. Podelementi se prikazuju samo u detaljnom pogledu (kada za svojstvo `ViewStyle` odredite vrednost `vsReport`) i kada su povezani sa pločicama svojstva `Columns`.

SLIKA 5.16 *ListView Item Editor*

U mom primeru RefList (jednostavna lista referenci na knjige, časopise, CD-ROM-ove i web sajtove) elementi se čuvaju u fajlu, jer korisnici programa mogu editovati sadržaj liste, koja se automatski čuva kada se izađe iz programa. Na ovaj način promene koje izvrši korisnik postaju trajne.

Čuvanje i učitavanje sadržaja ListView nije trivijalno jer tip `TListItems` nema automatski mehanizam čuvanja podataka. Drugi jednostavan pristup koji sam upotrebio je kopiranje podataka u listu stringova i iz nje, koristeći korisnički format. Lista stringova se zatim može čuvati u fajlu i ponovo učitati jednom komandom.

Format fajla je jednostavan, kao što možete videti u narednom sačuvanom kodu. Za svaki element liste program čuva naslov u jednoj liniji, indeks slike u drugoj (na početku linije je karakter @), a podelemente u narednim linijama koje su uvučene karakterom tabulatora:

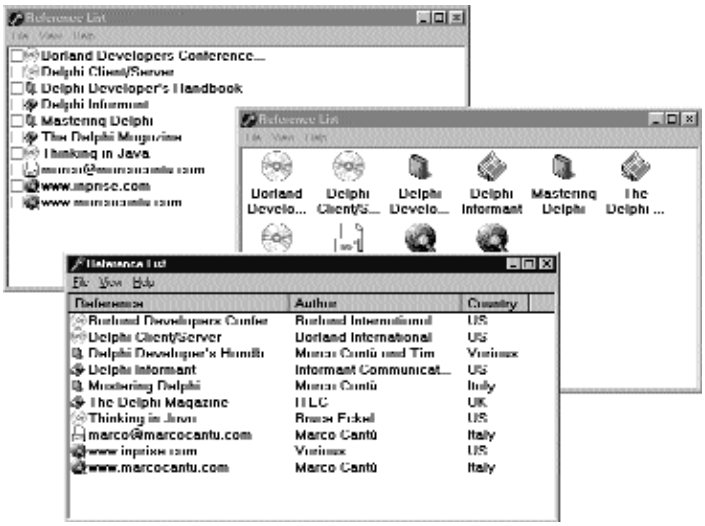
```
procedure TForm1.FormDestroy(Sender: TObject);
var
  I, J: Integer;
  List: TStringList;
begin
  // store the items
  List := TStringList.Create;
  try
    for I := 0 to ListView1.Items.Count - 1 do
```

```
    begin
      // save the caption
      List.Add (ListView1.Items[I].Caption);
      // save the index
      List.Add ('@' + IntToStr (ListView1.Items[I].ImageIndex));
      // save the subitems (indented)
      for J := 0 to ListView1.Items[I].SubItems.Count - 1 do
        List.Add (#9 + ListView1.Items[I].Subltems [J]);
    end;
    List.SaveToFile (
      ExtractFilePath (Application.ExeName) + 'Items.txt');
  finally
    List.Free;
  end;
end;
```

Elementi se zatim ponovo učitavaju u drugom delu metoda `FormCreate`:

```
procedure TForm1.FormCreate(Sender: T0bject);
var
  List: TStringList;
  NewItem: TListItem;
  I: Integer;
begin
  ...
  // load the items
  ListView1. Items.Clear;
  List := TStringList.Create;
  try
    List.LoadFromFile (
      ExtractFilePath (Application.ExeName) + 'Items.txt');
    for I := 0 to List.Count —1 do
      if List [I][1] = #9 then
        NewItem.SubItems.Add (Trim (List [I]))
      else if List [I][1] = '@' then
        NewItem.ImageIndex := StrToIntDef (List [I][2], 0)
      else
      begin
        // a new item
        NewItem ListView1.Items.Add
        NewItem.Caption := List [I];
     end;
  finally
    List.Free;
  end;
end;
```

Program sadrži meni koji možete upotrebiti da biste odabrali jedan od dva podržana pogleda kontrolom ListView, i da elementima dodate polja za potvrdu, kao kod CheckedListBoxa. Možete videti neke od različitih kombinacija ovih stilova na slici 5.17.



SLIKA 5.17 *Različiti primeri izlaza komponente ListView programa RefList dobijenih promenom svojstva ViewStyle i dodavanjem polja za potvrdu*

Još jedna važna karakteristika, koja je uobičajena za detaljan pogled ili izvešaj kontrole, je da prepustite korisniku da sortira elemente jedne od kolona. Da bismo ovo postigli, potrebno je izvršiti tri operacije. Prva je da za svojstvo `SortType` kontrole ListView dodelimo vrednost `stBoth` ili `stData`. Na ovaj način ListView će izvršiti sortiranje ne prema naslovima već pozivom događaja `OnCompare` za svaki par elemenata koje treba da sortira. Pošto je potrebno da sortiranje izvršimo za svaku od kolona u detaljnom pogledu, obradićemo događaj `OnColumnClick` (koji se odigrava kada korisnik klikne naslove kolona u detaljnom pogledu, ali samo ukoliko je svojstvu `ShowColumnHeader` dodeljena vrednost `True`). Svaki put kada se klikne kolona, program čuva broj kolone u privatnom polju `nSortCol` klase formulara:

```
procedure TForm1.ListView1ColumnClick(Sender: TObject;
  Column: TListColumn);
begin
  nSortCol := Column.Index;
  ListView1.Al haSort;
end;
```

Zatim, u trećem koraku, kod sortiranja koristi ili naslov ili jedan od podelemenata prema koloni sortiranja:

```
procedure TForm1.ListView1Compare(Sender: TObject;
  Item1, Item2: TListItem;
  Data: Integer; var Compare: Integer);
begin
  if nSortCol = 0 then
    Compare := CompareStr (Item1.Caption, Item2.Caption)
  else
    Compare := CompareStr (Item1.SubItems [nSortCol - 1],
      Item2.Subltems [nSortCol - 1]);
end;
```

Završne karakteristike koje sam dodao programu se odnose na operacije mišem. Kada korisnik klikne element levim tasterom miša, program RefList prikazuje opis selektovanog elementa. Kada klikne selektovani element desnim tasterom miša, prebacuje element u mod editovanja i korisnik može promeniti element (imajte na umu da će se izmene automatski sačuvati kada program prekine izvršavanje). Evo koda za obe operacije u obradi događaja OnMouseDown kontrole ListView:

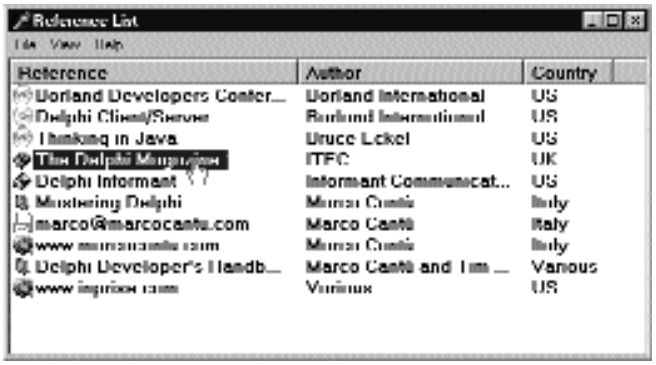
```
procedure TForm1.ListView1MouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
var
  strDescr: string;
  I: Integer;
begin
  // if there is a selected item
  if ListView1.Selected <> nil then
  if Button = mbLeft then
  begin
    // create and show a description
    strDescr := ListView1.Columns [0].Caption + #9 +
      ListView1.Selected.Caption + #13;
    for I := 1 to ListView1.Selected.Subltems.Count do
      strDescr := strDescr + ListView1.Columns [I].Caption + #9 +
        ListView1.Selected.Subltems [I-1] + #13;
    ShowMessage (strDescr);
    end;
    else if Button = mbRight then
      // edit the caption
      ListView1 .Selected.EditCaption;
end;
```

Mada ovaj primer nije kompletan, ipak prikazuje neke mogućnosti kontrole ListView. Ja sam, takođe, aktivirao karakteristiku "hot-tracking" Windowsa 98 koja omogućava da se element istakne i podvuče kada se nađe ispod pokazivača miša kao što možete videti na slici 5.18. Relevantna svojstva kontrole ListView se mogu pogledati u njenom tekstualnom opisu:

```
object ListView1: TListView
  Align = alClient
  Columns = <
    item
      Caption = 'Reference'
      Width = 230
    end
    item
      Caption = 'Author'
      Width =180
    end
    item
      Caption = 'Country'
      Width = 80
    end>
  Font.Height = -13
  Font.Name ='MS Sans Serif'
  Font.Style = [fsBold]
  FullDrag = True
  HideSelection = False
```

```
    HotTrack = True
    HotTrackStyles = [htHandPoint, htUnderlineHot]
    SortType = stBoth
    ViewStyle = vsList
    OnColumnClick = ListViewlColumnClick
    OnCompare = ListView1Compare
    OnMouseDown = ListView1MouseDown
  end
```

Ovaj program je prilično zanimljiv, a proširiću ga u Poglavlju 8 dodajući mu okvir za dijalog.

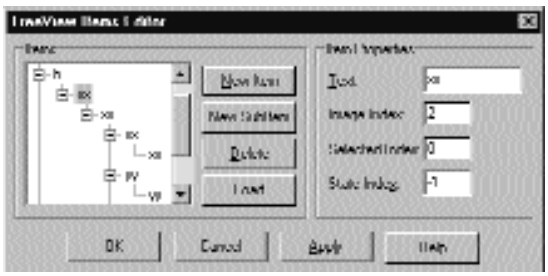


SLIKA 5.18 *Nova hot-tracking karakteristika kontrole ListView. Primetićete da su elementi sortirani prema autoru.*

Drvo podataka

Do sada smo videli primer koji koristi kontrolu ListView pa poglavlje možemo zaključiti kontrolom TreeView. Kontrola TreeView sadrži korisnički interfejs koji je fleksibilan i moćan (sa podrškom za editovanje i prevlačenje elemenata). Ona, takođe, predstavlja standard jer je deo korisničkog interfejsa Windows Explorera. Postoji veliki broj svojstava i načina prilagođavanja bitmape svake linije ili svakog tipa linije.

Da biste definisali strukturu čvorova kontrole TreeView u vreme dizajniranja, možete upotrebiti editor svojstava TreeView `Items` (pogledajte sliku 5.19). U ovom slučaju sam odlučio da učitam u TreeView podatke na početku izvršavanja, na način sličan prethodnom primeru.



SLIKA 5.19 *Editor svojstava TreeView Items*

Svojstvo `Items` komponente TreeView sadrži mnoge funkcije članice koje možete upotrebiti da biste izmenili hijerarhiju stringova. Na primer, možemo kreirati drvo sa dva nivoa narednim linijama:

```
var
  Node: TTreeNode;
begin
  Node := TreeView1.Items.Add (nil, 'First level');
  TreeView1.Items.AddChild (Node, 'Second level');
```

Upotrebom ova dva metoda (`Add` i `AddChild`) možemo kreirati kompleksnu strukturu u vreme izvršavanja. Ali, kako da učitamo informaciju? Ponovimo, možete upotrebiti StringList u vreme izvršavanja, učitati informacije iz tekst fajla i interpretirati ih.

Ipak, pošto kontrola TreeView sadrži metod `LoadFromFile`, primer DragTree koristi sledeći jednostavan kod:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  TreeView1.LoadFromFile (
    ExtractFilePath (Application.EXeName) + 'TreeText.txt');
end;
```

Metod `LoadFromFile` u osnovi učitava podatke u listu stringova i proverava nivo svakog elementa prebrojavanjem karaktera tabulatora. (Ukoliko ste radoznali, pogledaje metod `TTreeStrings.GetBufStart`, koji možete pronaći u jedinici ComCtl VCL izvornog koda koji dobijate uz Delphi.) Pomenimo još da su podaci koje sam pripremio za TreeView organizaciona šema multinacionalne kompanije.

Pored učitavanja podataka, program ih čuva kada završi izvršavanje, čineći time izmene trajnim. On sadrži i nekoliko elemenata menija za prilagođavanje fonta kontrole TreeView i za promenu još nekih jednostavnih vrednosti. Specifična karakteristika koju sam implementirao u ovaj primer je podrška za prevlačenje elemenata i celokupnih podstabala. Ja sam za svojstvo `DragMode` odredio vrednost `dmAutomatic` i napisao sam obrade događaja za `OnDragOver` i `OnDragDrop`.

U prvoj od dve obrade program proverava da li korisnik pokušava da prevuče element preko podelementa (koji bi se premestio uz element, što bi dovelo do beskonačne rekurzije):

```
procedure TForm1.TreeView1DragOver(Sender, Source: TObject;
  X, Y: Integer; State: TDragState; var Accept: Boolean);
var
  TargetNode, SourceNode: TTreeNode;
begin
    TargetNode := TreeView1.GetNodeAt (X, Y);
    // accept dragging from itself
  if (Source = Sender) and (TargetNode <> nil) then
  begin
    Accept := True;
    // determines source and target
    SourceNode = TreeView1. Selected;
    // look up the target parent chain
    while (TargetNode.Parent <> nil) and
      (TargetNode <> SourceNode) do
    TargetNode := TargetNode.Parent;
    // if source is found
```

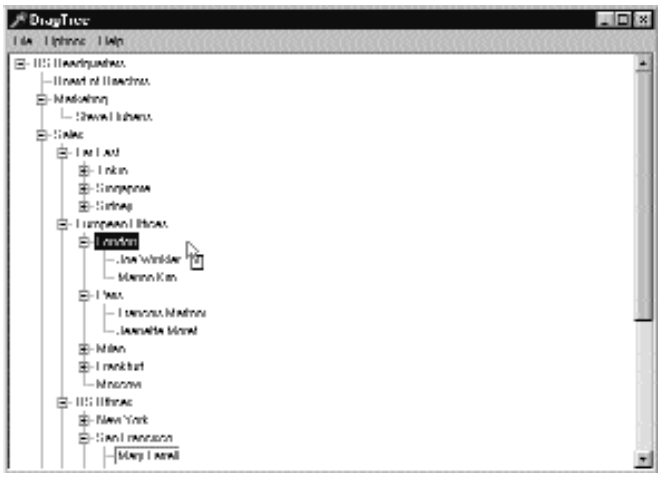
```
    if TargetNode = SourceNode then
      // do not allow dragging over a child
      Accept := False;
  end
  else
    Accept := False;
end;
```

Efekat ovog koda je (izuzev specifičnog slučaja koji smo morali da onemogućimo) da korisnik može prevući element kontrole TreeView preko drugog elementa, kao što je pokazano slikom 5.20. Pisanje koda za premeštanje je zaista jednostavno, jer kontrola TreeView obezbeđuje podršku za ovu operaciju putem metoda `MoveTo` klase `TTreeNode`:

```
procedure TForm1.TreeView1DragDrop(Sender,
  Source: TObject; X, Y: Integer);
var
  TargetNode, SourceNode: TTreeNode;
begin
  TargetNode := TreeView1.GetNodeAt (X, Y);
  if TargetNode <> nil then
  begin
    SourceNode := TreeView1.Selected;
    SourceNode.MoveTo (TargetNode, naAddChildFirst);
    TargetNode.Expand (False);
    TreeView1.Selected := TargetNode;
  end;
end;
```

NAPOMENA

Među primerima u direktorijumu Demos koje dobijate uz Delphi, postoji jedan interesantan primer koji prikazuje iscrtavanje kontrole TreeView. Primer se nalazi u poddirektorijumu CustomDraw. ■



SLIKA 5.20 *Primer DragTree kada je u toku operacija prevlačenja*

Šta je sledeće?

U ovom poglavlju smo počeli da istražujemo neke osnovne komponente koje su na raspolaganju u Delphiju. Ove komponente odgovaraju standardnim Windows kontrolama i nekim od Win32 kontrola i veoma su česte u aplikacijama. Takođe smo videli kako da kreiramo glavne menije i menije, a videli smo i kako da dodamo grafiku nekim od ovih kontrola.

Takođe smo istražili veoma važnu i još uvek malo korišćenu komponentu ActionList i njenu arhitekturu za obradu događaja elemenata menija i kontrola palete alata. Na ovu temu ćemo se vratiti u drugim primerima, a standardne MDI akcije i akcije nad skupom podataka ćemo prikazati u odgovarajućim poglavljima.

Naredni korak je detaljno upoznavanje sa najčešćim elementima Delphi programiranja — formularima. Već smo mnogo puta koristili formulare, ali je još uvek ostalo mnogo novih karakteristika koje treba obraditi, uključujući i neke važne karakteristike.

POGLAVLJE

6

Formulari, prozori i aplikacije

Ukoliko ste pročitali prethodno poglavlje, trebalo bi da sada možete da koristite standardne komponente Delphija u Vašim aplikacijama. Dakle, posvetimo sada pažnju centralnom elementu u razvoju Delphija: formularu. Formulare smo koristili još od početnih poglavlja, ali ja nikada nisam detaljno opisao šta možete učiniti pomoću njih, koja svojstva možete upotrebiti ili koji metodi klase TForm su naročito interesantni.

Ovo poglavlje se bavi nekim svojstvima i stilovima formulara, kao i određivanjem njihove veličine i pozicioniranjem. Takođe ću predstaviti aplikacije koje sadrže više formulara i posvetiću pažnju globalnim VCL objektima koji obrađuju interakcije među formularima, objektima `Screen` i `Application`. Posvetiću malo pažnje i unosu sa formulara, kako pomoću tastature, tako i pomoću miša. Dozvolite mi da ovo poglavlje počnem opštom, teorijskom diskusijom o formularima i prozorima.

Formulari nasuprot prozorima

U Windowsu većinu elemenata korisničkog interfejsa čine prozori. Zbog toga je i većina Delphi komponenata bazirana na prozorima — većina njih, ali ne sve komponente. Naravno, to nije ono što korisnik vidi. Razlika nije očigledna tako da naredne definicije morate pažljivo razmotriti. Zatim ćemo moći dalje da istražujemo.

- Sa korisničke tačke gledišta, prozor je deo ekrana okružen bordurom (granicom), koji sadrži naslov i obično sistemski meni, može se pomerati po ekranu, zatvoriti, a može se i smanjiti i prikazati preko celog ekrana. Prozori se mogu pomerati po ekranu ili unutar drugih prozora, kao kod MDI aplikacija (Multiple Document Interface — interfejs za više dokumenata). Ovi korisnički prozori se mogu podeliti u dve opšte kategorije. To su glavni prozori i okviri za dijalog.
- Tehnički govoreći, prozor je stavka interne memorije Windows sistema, koji često odgovara elementu koji je vidljiv na ekranu i uz koji je pridružen neki kod. Jedna od Windowsovih sistemskih biblioteka sadrži spisak svih prozora koje je kreirala svaka od aplikacija i svakom od njih dodeljuje jedinstveni broj (koji se obično naziva hendl — handle). Neke od ovih prozora korisnik vidi kao takve (kao prozore — pogledajte prvu definiciju), drugi imaju ulogu kontrola ili vizuelnih komponenata, neke privremeno kreira sistem (na primer, za prikazivanje menija), dok ostale kreira aplikacija, ali oni ostaju skriveni od korisnika.

Zajednička karakteristika svih prozora je da Windows sistem zna da postoje i da pozivaju funkciju za svoje ponašanje; svaki put kada se nešto dogodi u sistemu, šalje se poruka odgovarajućem prozoru koji odgovara izvršavanjem nekog koda. Svaki prozor sistema, zapravo, sadrži odgovarajuću funkciju (poznatu pod nazivom procedura prozora — window procedure), koja obrađuje različite poruke koje se tiču prozora.

U Delphi aplikaciji sistem konvertuje ove poruke niskog nivoa u događaje. Međutim, ponekad, što smo već videli u nekim primerima, poruke niskog nivoa obrađujemo direktno u formularu. Delphi nam omogućava da radimo na višem nivou od sistema čineći razvoj aplikacija lakšim.

NAPOMENA

Memorijska oblast u Windows sistemu koja je alocirana za spisak svih prozora koji su kreirani je ograničena. Kreiranje previše prozora smanjuje takozvane sistemske resurse. Windows 3.1 je imao veliko ograničenje kad je u pitanju broj prozora koji su mogući na sistemu. Pod Windowsom 95 i 98 ovaj limit je prilično povećan, a pod Windowsom NT limit ne postoji. Kada se desi da u sistemu ima previše prozora (uključujući sve kontrole i sakrivene prozore), ne možete kreirati još jedan prozor, što je situacija koja će blokirati većinu aplikacija. Zbog toga u Delphiju postoji veliki broj komponenata za koje nije potreban prozor, uključujući oznake. Ovakav pristup Vam omogućava da sačuvate prilično količinu sistemske memorije, a da ne morate da brinete (ili da znate) o tome. Kao što je već pomenuto u Poglavlju 4, grafičke kontrole koje nisu u prozorima takođe imaju i druge prednosti, uključujući i brže kreiranje i iscrtavanje, a i manje zahtevaju. ■

Imajući na umu ove opšte definicije, možemo da se vratimo na Delphi i da pokušamo da shvatimo ulogu formulara. Formulari predstavljaju prozore sa korisničke tačke gledišta i mogu se koristiti za kreiranje glavnih prozora, MDI prozora i okvira za dijalog. Njihovo ponašanje je definisano uglavnom kodom koji je za njih napisan, ali to ponašanje se određuje i nekim veoma važnim svojstvima, `FormStyle` i `BorderStyle`, koja ćemo kratko opisati. Mnoge druge komponente su zasnovane na prozorima, ali za korisnika samo formulari izgledaju kao prozori. Ostale komponente u okviru prozora, ili kontrole, se mogu smatrati prozorima samo prema njihovoj definiciji.

Kao primer, jednostavno kreirajte novu aplikaciju i smestite kontrolu, sačuvajte fajlove u direktorijumu sa njihovim unapred određenim nazivima i pokrenite program. Upotrebom alata WinSight koji se nalazi u okviru Delphija možete videti spisak prozora sistema; obratite naročito pažnju na prozore koje kreira aplikacija, kao što je prikazano na slici 6.1. Među njima se nalaze i sledeći prozori:

- Glavni prozor, formular, sa naslovom *Form1*. Ovaj formular predstavlja prozor klase `TForm1`.
- Pripadajući prozor sa kontrolom unutar formulara, čiji je naziv *Button1*. Ovo je pripadajući prozor klase `TButton`.
- Sakriveni glavni prozor, prozor aplikacije, nazvan *Project1*. Ovo je prozor klase `TApplication`.

Primetićete da se nazivi u uglastim zagradama koji se prikazuju u WinSightu, koji predstavljaju interne nazive sistema, odgovaraju nazivima klasa Delphi komponenata.

SLIKA 6.1 *Prozori jednostavne aplikacije onako kako se prikazuju u WinSightu*

PREKLOPLJENI, ISKAČUĆI (POP-UP) I PRIPADAJUĆI PROZORI

Da biste razumeli uloge različitih prozora ovog programa, potrebno je da se pozabavimo nekim tehničkim elementima koji se odnose na Windows okruženje. To nisu jednostavni koncepti, ali vredi da ih upoznate.

Svaki prozor koji kreirate sadrži tri opšta stila koja određuju njegovo ponašanje. Ti stilovi su preklapanje, pop-up i pripadanje:

- Prozori koji se preklapaju (overlapped) su glavni prozori aplikacije, koji se ponašaju onako kako biste to od njih očekivali.
- Pop-up prozori se često koriste za okvire za dijalog i poruke, a mogu se smatrati prozorima koji su nasleđeni iz prethodnih verzija sistema. Zapravo, u Windowsu 1 prozori se nisu preklapali već su se prikazivali jedan uz drugi, a samo pop-up prozori su mogli da prekriju druge prozore. Pop-up prozori su veoma slični prozorima koji se preklapaju.
- Treća grupa, pripadajući prozori (child), su se prvobitno koristili za kontrole unutar okvira za dijalog. Vi možete da koristite ovaj stil za bilo koji prozor koji se ne može pomeriti van klijent površine roditeljskog prozora. Očigledno, proširenje je upotreba pripadajućih prozora za izradu MDI aplikacija; međutim, ovo ponašanje nije automatsko.

Veoma je važno da primetite da samo, tehnički govoreći, pripadajući prozori mogu imati roditeljske prozore. Bilo koji drugi prozor može imati vlasnika. Vlasnik je prozor koji ima neprekidnu razmenu poruka sa prozorima koje sadrži — na primer, kada se prozor redukuje na ikonu, kada se aktivira i tako dalje. Pripadajući prozori ne koriste ekranske koordinate; umesto toga ovakvi prozori koriste klijent koordinate svojih roditeljskih prozora — da bi se prikazali, pozajmljuju piksele ne sa ekrana već od svog roditeljskog prozora.

Primetite da Windows API koristi isti termin (parent — roditelj) da nazanači i roditelja i vlasnika. Čak i API funkcija `GetParent` može kao rezultat dati oba elementa. U okviru sistema, dve veze (veza roditeljskog prozora i veza prozora vlasnika) se čuvaju odvojeno. To je zaista veoma čudno i dovodi do zabune.

U Delphiju su svi formulari preklopljeni prozori, uključujući i okvire za dijalog, a formular je vlasnik svih komponenata u okviru prozora (kontrola) koje smestite na formular. Ipak, njhov roditej može biti formular ili neka od specijalnih kontejnerskih komponenata, kao što je, recimo, GroupBox ili Panel. Kada unutar GroupBoxa smestite opcionu kontrolu, GroupBox je roditelj, ali je vlasnik formular. A kod pop-up prozora? U Delphiju se oni koriste za skrivene prozore aplikacije, liste za combo polja i za prozore sa savetima. U sistemu se koriste za poruke i padajuće ili iskačuće menije, tek da pomenem dve primene.

Svojstvo `Parent` kontrole naznačava ko je odgovoran za njeno prikazivanje. Kada prevučete komponentu na formular prilikom dizajniranja, formular će postati i vlasnik i roditelj. Kada kontrolu kreirate u vreme izvršavanja, biće potrebno da odredite vlasnika (koristeći parametar konstruktora Create), ali takođe morate da odredite svojstvo Parent, ili kontrola neće biti vidljiva.

Aplikacija je prozor

Analizom informacije WinSignt možda ste primetili da program sadrži dodatni prozor za aplikaciju. Slično, u poslednjem poglavlju smo videli da elementi, dodati sistemskom meniju formulara, nisu dodati ikoni na Taskbaru. Prozor aplikacije je sakriven, ali se prikazuje na Taskbaru. Zbog toga Delphi naziva prozor *Form1*, a odgovarajuću Taskbar ikonu *Project1*.

Prozor koji odgovara `Application` objektu — prozor aplikacija — služi da okupi zajedno sve prozore aplikacije. Činjenica da svi formulari najvišeg nivoa programa imaju svoj skriveni prozor, na primer, je fundamentalna kada se aplikacija aktivira. Zapravo, kada su prozori Vašeg programa iza drugih prozora, klik na jedan od prozora aplikacije će sve prozore aplikacije premestiti u prvi plan. Drugim rečima, sakriveni prozor aplikacije se koristi za povezivanje različitih formulara aplikacije. Prozor aplikacije zapravo nije sakriven jer bi to uticalo na njegovo ponašanje; on jednostavno ima visinu i širinu nula, te zbog toga nije vidljiv.

Kada kreirate novu, praznu aplikaciju, Delphi generiše kod za fajl projekta koji sadrži sledeće:

```
begin
  Application.Initialize;
  Application.CreateForm (TForm1, Form1);
  Application.Run;
end;
```

Ovaj kod koristi globalni objekat `Application` koji je klase `Tapplication`, a definisan je VCL-om u jedinici Forms. Ovaj objekat je zaista komponenta mada ne možete da odredite njegova svojstva koristeći Object Inspector. Svojstva uključuju naziv izvršnog fajla (`ExeName`), naziv (`Title`) aplikacije (unapred je određeno da je to naziv izvršnog fajla bez ekstenzije), i ikonu (`Icon`) koja se prikazuje na Taskbaru. `Title` aplikacije možete videti na Windows Taskbaru. Isti naziv se prikazuje kada prelazite iz jedne u drugu aktivnu aplikaciju kombinacijom tastera Alt+Tab. Da biste izbegli razliku između ova dva naslova, naziv aplikacije možete promeniti u vreme dizajniranja, na strani Application okvira za dijalog Project Options, ili to možete učiniti u vreme izvršavanja tako što ćete kopirati naslov u Title aplikacije sledećim kodom:

```
Application.Title := Form1.Caption;
```

Možete, takođe, odrediti i druga svojstva globalnog objekta `Application` koristeći isti okvir za dijalog. Da bi se obradili događaji objekta `Application`, do Delphija 4 ste morali sami da napišete kod. Delphi 5, umesto toga, sadrži novu komponentu `ApplicationEvents`, specijalno namenjenu obradi događaja objekta Application. Pored lakšeg povezivanja obrade događaja u vreme izvršavanja, prednost upotrebe nove komponente leži u činjenici da dozovljava višestruke obrade. Ukoliko jednostavno postavite dve instance komponente ApplicationEvents u dva različita formulara, svaki od njih može obraditi isti događaj, a obe obrade će biti izvršene. Drugim rečima, više komponenata ApplicationEvents objekata može da poveže obrade u lanac.

Neki od ovih događaja na nivou aplikacije, uključujući `OnActivate`, `OnDeactivate`, `OnMinimize` i `OnRestore`, Vam omogućavaju da pratite status aplikacije. Ostali događaji se prosleđuju aplikaciji preko kontrola koje ih dobijaju, kao što su događaji `OnActionExecute`, `OnActionUpdate`, `OnHelp`, `OnHint`, `OnShortCut` i `OnShowHint`. Konačno, postoji i globalna obrada izuzetka `OnException` koju smo koristili u Poglavlju 3, događaj `OnIdle` koji smo koristili za izračunavanje u pozadini i obradu događaja `OnMessage` koji se odvija svaki put kada se pošalje poruka bilo kom prozoru ili kontroli u okviru prozora aplikacije.

U većini aplikacija ne morate brinuti o prozoru aplikacije, izuzev o određivanju svojstva `Title` i njegovoj ikoni, i o obradi nekih događaja. U svakom slučaju postoje neke operacije koje možete da obavite. Određivanje vrednosti `False` za svojstvo `ShowMainForm` u izvornom kodu projekta označava da glavni formular projekta ne treba prikazati na početku izvršavanja. Unutar programa možete upotrebiti svojstvo `MainForm` objekta `Application` da biste pristupili glavnom formularu, a to je prvi element koji kreira program.

Prikazivanje prozora aplikacije

Ne postoji bolji dokaz da zaista postoji prozor za objekat `Application`, nego da taj prozor prikažemo. Zapravo i nije potrebno da ga prikažemo — potrebno je samo da mu promenimo veličinu i da odredimo nekoliko atributa prozora, kao što su postojanje naslova i bordura. Ove operacije možemo da obavimo upotrebom Windows API funkcija nad prozorom koji je naznačen svojstvom `Handle` objekta `Application`:

```
procedure TForm1.Button1Click (Sender: TObject);
var
  OldStyle: Integer;
begin
  // add border and caption to the AppBrowser window
  OldStyle : = GetWindowLong (
    Application.Handle, gwl_Style);
  SetWindowLong (Application.Handle, gwl_Style,
    OldStyle or ws_ThickFrame or ws_Caption);
  // set the size of the AppBrowser window
  SetWindowPos (Application.Handle,
    0, 0, 0, 200, 100, swp_NoMove or swp_NoZOrder);
end;
```

Dve API funkcije, `GetWindowLong` i `SetWindowLong`, se koriste za pristup sistemskim informacijama vezanim za prozor. U ovom slučaju koristimo parametar `gwl_Style` da bismo pročitali ili zapisali stilove prozora, koji uključuju njegovu borduru, naslov, sistemski meni, ikone bordure i tako dalje. Gornji kod daje trenutne stilove i dodaje formularu standardnu borduru i zaglavlje (upotrebom iskaza or). Kao što ćemo videti kasnije u ovom poglavlju, često će biti potrebno da koristite ove API funkcije niskog nivoa u Delphiju jer postoje svojstva klase `TForm` koja daju isti efekat. Ovaj kod nam je potreban jer prozor aplikacije nije formular.

Izvršavanje ovog prozora prikazuje prozor projekta kao što možete da vidite na slici 6.2. Mada nije potrebno da implementirate nešto slično ovome u Vaše programe, izvršavanje ovog programa će otkriti vezu između prozora aplikacije i glavnog prozora Delphi programa. Ovo je veoma važno polazište ukoliko želite da razumete internu strukturu Delphi aplikacija.

SLIKA 6.2 *Sakriveni prozor aplikacije koji otkriva program ShowApp*

SAVET

U Windowsu su operacijama Maximize i Minimize po definiciji pridruženi sistemski zvuci i animirani vizuelni efekti. Aplikacije izrađene u verzijama Delphija do verzije 4 nisu davale zvuk ili nisu prikazivale vizuelne efekte (izuzev ukoliko niste napisali specifični kod). Delphi 5 aplikacije proizvode zvuk i po definiciji prikazuju vizuelne efekte. Jednostavno ponovo kompajlirajte Vaše programe i oni će dobiti ovu dodatnu karakteristiku! Razlog zbog koga ovoga nije bilo u ranijim verzijama je to što se sistemske poruke Minimize i Maximize nisu prosleđivale unapred određenoj proceduri prozora, gde Windows implementira sistemske zvuke, da bi se izbegli neželjeni efekti animacije na Taskbaru. Kako je pronađeno rešenje ovog problema u Delphiju 5, unapred određeno ponašanje je restaurirano prosleđivanjem poruke operativnom sistemu. ■

Sistemski meni aplikacije

Izuzev ukoliko ne napišete veoma čudan kod, kao što je kod aplikacije koju smo upravo videli, korisnici će videti prozor aplikacije samo na Taskbaru. Sa tog mesta možete aktivirati sistemski meni prozora tako što ćete kliknuti desnim tasterom miša. Kao što sam pomenuo kada sam razmatrao sistemski meni u prethodnom poglavlju, meni aplikacije nije isti kao sistemski meni glavnog formulara. U primeru SysMenu u Poglavlju 5 ja sam dodao sopstvene elemente sistemskom meniju glavnog formulara. Sada u primeru SysMenu2 želim da prilagodim sistemski meni prozora aplikacije na Taskbaru.

Prvo moramo da dodamo nove elemente sistemskom meniju prozora aplikacije kada se program pokrene. Evo ažuriranog koda metoda `FormCreate`:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  // add a separator and a menu item to the system menu
  AppendMenu (GetSystemMenu (Handle, FALSE),
    MF_SEPARATOR, 0, '');
  AppendMenu (GetSystemMenu (Handle, FALSE),
    MF_STRING, idSysAbout, '&About. . .');
  // add the same items to the application system menu
  AppendMenu (GetSystemMenu (Application. Handle, FALSE),
    MF_SEPARATOR, 0, 71;
  AppendMenu (GetSystemMenu (Application.Handle, FALSE),
    MF_STRING, idSysAbout, '&About...');
end;
```

Prvi deo koda dodaje separator i element sistemskom meniju glavnog formulara. Preostala dva poziva dodaju ista dva elementa sistemskom meniju aplikacije jednostavnim referisanjem na `Application.Handle`. Ovo je dovoljno da se prikaže ažurirani sistemski meni kao što možete videti kada pokrenete program. Sledeći korak je obrada izbora novog elementa menija.

Da bismo obradili poruke formulara, možemo jednostavno da napišemo nove obrade događaja ili metode za obrade događaja. Istu stvar ne možemo učiniti sa prozorom aplikacije jer je nasleđivanje od klase `TApplication` prilično složeno. Mnogo češće možemo jednostavno da napišemo obradu `OnMessage` događaja ove klase koji se aktivira svakom porukom iz reda poruka koju dobija aplikacija.

Da biste obradili događaj `OnMessage` globalnog objekta `Application`, jednostavno dodajte komponentu ApplicationEvents glavnom formularu i definišite obradu `OnMessage` događaja ove komponente. U ovom slučaju nam je potrebna obrada poruke `vw_SysCommand`, a obradu je potrebno sprovesti samo ukoliko parametar `wParam` označava da je korisnik odabrao element menija koji smo upravo dodali, element `idSysAbout`:

```
procedure TForm1.ApplicatiOnEvents1Messaqe(var Msg: tagMSG;
  var Handled: Boolean);
begin
  if (Msg.Message = wm_ysCommand) and
    (Msg.wParam = idSysAbout) then
  begin
    ShowMessage ('Mastering Delphi: SysMenu2 example');
    Handled : True;
  end;
end;
```

Ovaj metod je veoma sličan metodu koji je upotrebljen za obradu odgovarajućeg elementa sistemskog menija glavnog formulara:

```
procedure WMSysCommand (var Msg: TWMSysCommand);
  message-wmSysCommand;
...
procedure TForm1.WMSysCommand (var Msg: TWMSysCommand);
begin
  // handle a specific command
  if Msg.CmdType = idSysAbout then
    ShowMessage ('Mastering Delphi: SysMenu2 example');
  inherited;
end;
```

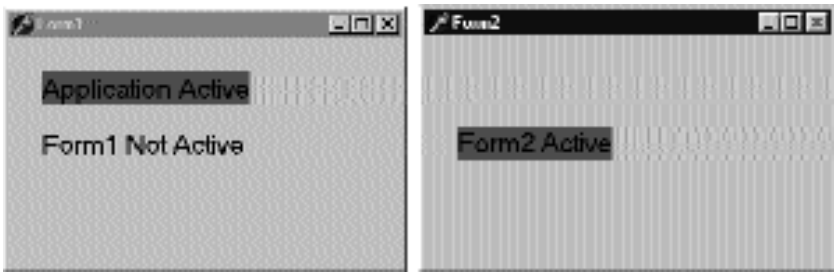
Aktiviranje aplikacija i formulara

Da bih pokazao kako funkcioniše aktiviranje aplikacija i formulara, napisao sam jednostavan primer koji sam nazvao ActivApp. Ovaj primer sadrži dva formulara. Svaki formular sadrži komponentu Label (`LabelForm`) koja se koristi za prikazivanje statusa formulara. Program za ovo koristi tekst i boju, kao što demonstriraju događaji `OnActivate` i `OnDeactivate` prvog formulara:

```
procedure TForm1.FormActivate(Sender: TObject);
begin
  LabelForm.Caption := 'Form2 Active';
  LabelForm.Color := clRed;
end;

procedure TForm1.FormDeactivate(Sender: TObject);
begin
  LabelForm.Caption := 'FormZ Not Active';
  Labelrorm.Color := clBtnFace;
end;
```

Drugi formular sadrži sličnu oznaku i sličan kod. Glavni formular takođe prikazuje status cele aplikacije. Glavni formular koristi komponentu ApplicationEvents za obradu događaja `OnActivate` i `OnDeactivate` objekta Application. Ove dve obrade događaja su slične dvema obradama koje su ranije prikazane, a jedina razlika je u tome što menjaju tekst i boju druge oznake formulara.

Ukoliko pokušate da pokrenete program, videćete da li je aplikacija aktivna i, ukoliko jeste, koji formulari su aktivni. Posmatranjem izlaza (pogledajte sliku 6.3) i osluškivanjem zvuka možete da razumete kako se svaki od događaja aktiviranja poziva u Delphiju. Pokrenite ovaj program i isprobajte ga sve dok ne budete razumeli kako funkcioniše. Ubrzo ćemo se vratiti na ostale događaje aktiviranja formulara.



SLIKA 6.3 *Primer ActivApp prikazuje da li je aplikacija aktivna i koji je od formulara aplikacije aktivan*

Određivanje stilova formulara i bordure

Među svojstvima formulara, dva svojstva određuju osnovna pravila ponašanja formulara: svojstva `FormStyle` i `BorderStyle`. Prvo od ova dva specijalna svojstva Vam omogućava da odaberete između normalnog SDI (Single Document Interface) i jednog od prozora koji čine MDI (Multiple Document Interface) aplikaciju.

Evo mogućih vrednosti svojstva FormStyle:

- `fsNormal`: Formular je normalan SDI prozor ili okvir za dijalog.
- `fsMDIChild`: Formular je pripadajući MDI prozor.
- `fsMDIForm`: Formular je MDI roditeljski prozor, to jest, prozor MDI aplikacije.

- `fsStayOnTop`: Formular je SDI prozor, ali uvek ostaje na vrhu svih drugih prozora izuzev ukoliko neki prozor ima karakteristitku ostani-na-vrhu (stayon-top).

Pošto su aplikaciji koja zadovoljava standard Multiple Document Interface potrebna dva prozora različitog tipa (okvir i dete), dve vrednosti svojstva `FormStyle` su uključene. Da biste izradili MDI aplikaciju, možete koristiti standardni šablon aplikacije ili pogledajte Poglavlje 8 koje detaljno opisuje MDI. Za sada može biti interesantno da istražite upotrebu stila `fsStayOnTop`.

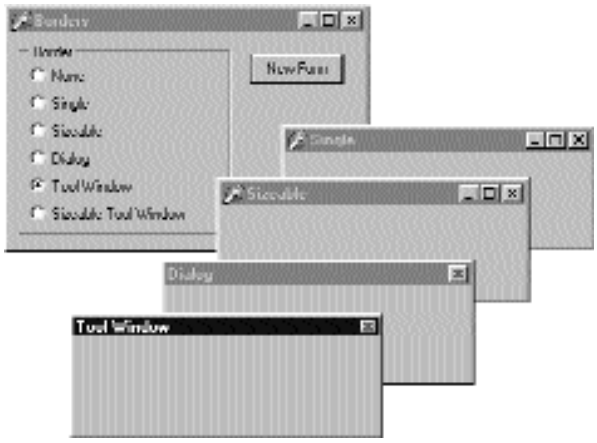
Da biste kreirali formular koji će biti na vrhu (formular čiji je prozor uvek na vrhu), potrebno je samo da odredite svojstvo `FormStyle`. Ovo svojstvo ima dva različita efekta u zavisnosti od vrste formulara na koji ga primenjujete:

- Glavni formular aplikacije će uvek biti ispred bilo koje druge aplikacije (izuzev ukoliko druga aplikacija ima isti stil).
- Sekundarni formular će ostati ispred bilo kog drugog formulara aplikacije kojoj pripada. Ovo se ne odnosi na prozore drugih aplikacija.

Stil bordure

Drugo svojstvo formulara je `BorderStyle`. Ovo svojstvo se odnosi na vizuelni element formulara, ali je mnogo uticajnije na ponašanje prozora, kao što možete videti sa slike 6.4.

U vreme dizajniranja formular se prikazuje sa unapred određenom vrednošću svojstva `BorderStyle`, a to je vrednost `bsSizable`. Ovo odgovara Windows stilu koji je poznat kao debeo okvir (thick frame). Kada glavni prozor ima debeo okvir, korisnik može da promeni veličinu prozora povlačenjem njegve bordure. Ovo je jasno naznačeno specijalnim kursorom *resize* (koji je u obliku dvostruke strelice) koji se prikazuje kada korisnik pomeri pokazivač miša nad debelu borduru prozora.



SLIKA 6.4 *Primeri formulara sa različitim stilovima bordure koje kreira primer Borders*

Druga važna vrednost ovog svojstva je `bsDialog`. Ukoliko je odaberete, formular će kao borduru imati tipični okvir okvira za dijalog — debeo okvir koji ne dozvoljava promenu veličine.

Primetićete da pored ovog grafičkog elementa, kada odaberete vrednost bsDialog, formular postaje okvir za dijalog. Ovo povlači brojne izmene. Na primer, elementi sistemskog menija su drugačiji, a formular će ignorisati neke elemente svojstva `BorderIcons`.

UPOZORENJE

Određivanje svojstva `BorderStyle` u vreme dizajniranja ne daje nikakav vizuelni efekat. Zapravo, nekoliko svojstva komponente ne daju nikakav efekat u vreme dizajniranja jer bi Vas sprečila u radu sa komponentom prilikom razvoja programa. Na primer, kako biste promenili veličinu formulara upotrebom miša kada biste formular pretvorili u okvir za dijalog? Kada pokrenete aplikaciju, formular će imati borduru kakvu ste naznačili. ■

Postoje još četiri vrednosti koje možete odabrati za svojstvo `BorderStyle`. Stil `bsSingle` se može upotrebiti za kreiranje glavnog prozora kojem se ne može menjati veličina. Mnoge igre i aplikacije se baziraju na prozorima sa kontrolama (recimo formulari za unos podataka) koji koriste ovu vrednost, jer inače promena veličine ne bi imala smisla. Promena veličine da bi se prikazala prazna površina, ili smanjivanje formulara čime neke kontrole ne bi bile vidljive, ne pomažu korisniku programa (mada automatski klizači u Delphiju praktično rešavaju ovaj problem). Vrednost `bsNone` se koristi u specijalnim situacijama i unutar drugih formulara. Nikada nećete videti aplikaciju u kojoj glavni prozor nema borduru ili zaglavlje (izuzev možda kao primer u knjizi o programiranju da bi Vam se pokazalo da to nema smisla).

Preostale dve vrednosti, `bsToolWindow` i `bsSizeToolWin`, se odnose na specifični Win32 stil `ws_ex_ToolWindow`. Ovaj stil pretvara prozor u pokretnu paletu alata sa malim fontom i kontrolom za zatvaranje. Ovaj stil ne bi trebalo koristiti za glavni prozor aplikacije.

Da bih testirao efekat i ponašanje različitih vrednosti svojstva `BorderStyle`, napisao sam jednostavan program koji sam nazvao Borders. Njegov izlaz ste već videli na slici 6.4. Ipak, savetujem Vam da pokrenete program i da eksperimentišete sa njim sve dok u potpunosti ne shvatite sve razlike između formulara.

Glavni formular ovog programa sadrži grupu opcionih kontrola i jednu kontrolu. Takođe postoji i sekundarni formular koji nema komponente, i za njegovo svojstvo `Position` je određena vrednost poDefaultPosOnly. Ovo se odnosi na početnu poziciju drugog formulara koji će biti kreiran kada kliknemo kontrolu. (Svojstvo Position ću razmatrati kasnije u ovom poglavlju.)

Kod programa je veoma jednostavan. Kada kliknete kontrolu, novi formular se kreira dinamički, prema već odabranom elementu grupe opcionih kontrola:

```
procedure TForm1.BtnNewFormClick (Sender: TObject);
var
  NewForm: TForm2;
begin
  NewForm := TForm2.Create (Application);
  NewForm.BorderStyle := TFormBorderStyle (
    BorderRadioGroup.ItemIndex);
  NewForm.Caption := BorderRadioGroup.Items[
    BorderRadioGroup. ItemIndex];
  NewForm. Show;
end;
```

Ovaj program zapravo koristi trik: on pretvara broj odabranog elementa u enumeraciju (nabrajanje) `TFormBorderStyle`. Ovo funkcioniše jer su elementi poređani po vrednosti ovog nabrajanja:

```
type
  TFormBorderStyle = (bsNone, bsSingle, bsSizeable,
    bsDialog, bsToolWindow, bsSizeTooWin);
```

Metod `BtnNewFormClick` zatim kopira tekst opcione kontrole u zaglavlje sekundarnog formulara. Ovaj program se poziva na `TForm2`, sekundarni formular definisan u sekundarnoj jedinici programa, a sačuvan je kao `SECOND.PAS`. Dakle, da biste kompajlirali primer, morate dodati sledeće linije u odeljak `implementation` jedinice glavnog formulara:

```
uses
  Second;
```

SAVET

Uvek kada je potrebno da se pozovete na neku drugu jedinicu programa, umetnite odgovarajuću liniju `uses` u odeljak `implementation` umesto u odeljak `interface` ukoliko je moguće. Na ovaj način ćete ubrzati proces kompajliranja, dobićete čistiji kod (jer su jedinice koje uključujete odvojene od jedinica koje uključuje Delphi) i nikada se ne generišu cirkularne reference između različitih jedinica. Da biste ovo postigli, takođe možete upotrebiti komandu File➥Use Unit. ■

Ikone bordure

Još jedan važan element formulara je postojanje ikona na njegovoj borduri. Po definiciji, fomular sadrži malu ikonu koja je dodeljena sistemskom meniju, kontrole Minimize, Maximize i Close na desnoj strani formulara. Različite opcije možete da odredite koristeći svojstvo `BorderIcons` za koje možete upotrebiti četiri moguće vrednosti: `biSystemMenu`, `biMinimize`, `biMaximize` i `biHelp`.

NAPOMENA

Ikona bordure `biHelp` aktivira pomoć "What's this?". Kada je uključen ovaj stil, a isključni su stilovi `biMinimize` i `biMaximize`, pojavljuje se znak pitanja na naslovnoj liniji formulara. Kada kliknete znak pitanja, a zatim kliknete komponentu formulara (ali ne na sam formular!), Delphi aktivira Help o objektu unutar iskačućeg prozora. Ovo je pokazano primerom BIcons koji sadrži jednostavan Help fajl sa stranom povezanom sa svojstvom HelpContext kontrole u sredini formulara. ■

Primer BIcons prikazuje ponašanje fomulara sa četiri različite ikone bordure i pokazije kako da ovo svojstvo promenite u vreme izvršavanja. Formular ovog primera je veoma jednostavan: sadrži samo meni koji sadrži četiri elementa, po jedan za svaki mogući element skupa ikona bordura. Ja sam napisao jedan metod, povezan sa četiri komande, koji čita oznake elemenata menija da bi se odredila vrednost svojstva `BorderIcons`. Stoga je ovaj kod dobra vežba za rad sa skupovima:

```
procedure TForm1.SetIcons(Sender: TObject);
var
  BorIco: TBorderIcons;
begin
  (Sender as TMenuItem).Checked :=
    not (Sender as TMenuItem).Checked;
  if SystemMenu1.Checked then
    BorIco := [biSystemMenu]
  else
    BorIco := [];
  if MaximizeBox1.Checked then
    Include (BorIco, biMaximize);
  if MinimizeBox1.Checked then
    Include (BorIco, biMinimize);
  if Help1.Checked then
    Include (BorIco, biHelp);
  BorderIcons := BorIco;
end;
```

Dok se izvršava primer BIcons, Vi lako možete da odredite i uklonite različite vizuelne elemente bordure formulara. Odmah ćete primetiti da su neki od ovih elemanta u bliskoj vezi: ukoliko uklonite sistemski meni, nestaće i sve ikone bordure; ukoliko uklonite bilo kontrolu Minimize bilo kontrolu Maximize, one će postati sive; ukoliko uklonite obe kontrole, one neće biti prikazane. Primetite da u poslednja dva slučaja odgovarajući elementi sistemskog menija nisu aktivni. Ovo je uobičajeno ponašanje za Windows aplikacije. Kada su kontrole Maximize i Minimize neaktivne, možete da aktivirate kontrolu Help. Kao prečicu za postizanje ovog efekta možete da kliknete kontrolu unutar formulara. Takođe, možete kliknuti kontrolu pošto ste kliknuli ikonu Help Menu da biste videli Help poruku, što je prikazano slikom 6.5.

SLIKA 6.5 *Kontrola Help koju prikazuje primer BIcons. Pomeranjem kursora Help iznad kontrole prikazaće se Help prikazan na slici.*

Dodatna funkcija programa je to što prikazuje vreme u zaglavlju kada je Help pozvan, a to se postiže obradom `OnHelp` događaja formulara. Ovaj efekat je vidljiv na slici.

Određivanje drugih stilova prozora

Stil bordure i ikone bordure su određene dvama različitm Delphi svojstvima koja se mogu koristiti za određivanje početne vrednosti odgovarajućih elemenata korisničkog interfejsa. Videli smo da, osim toga što menjaju izgled korisničkog interfejsa, ova svojstva menjaju i ponašanje prozora. Važno je znati da ova svojstva bordure i svojstvo `FormStyle` uglavnom odgovaraju različitim vrednostima stila (style) i proširenog stila (extended style) prozora. Ova dva termina se odnose na dva parametra API funkcije `CreateWindowEx` koju Delphi koristi za kreiranje formulara.

Ovo je veoma važno znati jer Vam Delphi omogućava da izmenite ova dva parametra zaobilazeći virtuelni metod `CreateParams`:

```
public
  procedure CreateParams (
    var Params: TCreateParams); override;
```

Ovo je jedini način da upotrebite neke čudne stilove prozora koji nisu direktno dostupni preko svojstava. Spisak stilova i proširenih stilova ćete pronaći u API Helpu pod temama *CreateWindow* i *CreateWindowEx*. Videćete da Win32 API sadrži veliki broj stilova za ove funkcije, uključujući i one koji se odnose na prozore sa alatima.

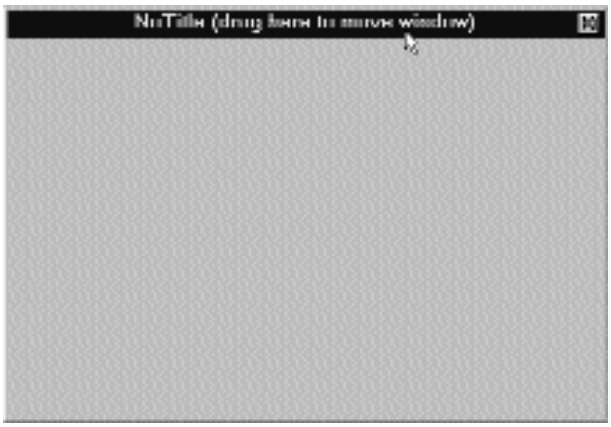
Da bih Vam pokazao kako da upotrebite ovaj pristup, napisao sam primer NoTitle koji Vam omogućava da kreirate program sa sosptvenim zaglavljem. Prvo moramo da uklonimo standardno zaglavlje, ali da usput zadržimo okvir za promenu veličine tako što ćemo odrediti odgovarajuće stilove:

```
procedure TForm1.CreateParams (var Params: TCreateParams);
begin
  inhereted CreateParams (Params);
  Params.Style := (Params.Style or ws_Popup) and
    not ws_Caption;
end;
```

NAPOMENA

Pored promene stila i drugih karakteristika prozora kada se on kreira, Vi ih možete promeniti u vreme izvršavanja mada neke karakteristike nemaju efekat. Da biste promenili većinu parametara kreiranja u vreme izvršavanja, možete upotrebiti API funkciju `SetWindowLong` koja Vam omogućava da promenite internu informaciju prozora. Odgovarajuća funkcija `GetWindowLong` se može koristiti za dobijanje trenutnog statusa. Druge dve funkcije, `GetClassLong` i `SetClassLong`, se mogu koristiti za čitanje i izmenu stilova klase (informacije strukture `WindowClass` klase `TCreateParams`). Često ćete koristiti ove Windows API funkcije niskog nivoa u Delphiju, izuzev ukoliko ne pišete napredne komponente. ■

Da bismo uklonili zaglavlje, potrebno je da promenimo stil preklapanja u pop-up stil, inače će zaglavlje ostati. Kako da dodamo sopstveno zaglavlje? Ja sam postavio oznaku poravnatu sa gornjom bordurom formulara i malom kontrolom na kraju. Ovaj efekat u vreme izvršvanja možete videti na slici 6.6.



SLIKA 6.6 *Primer NoTitle nema pravo zaglavlje, a lažno je postignuto oznakom*

Da bismo učinili da lažno zaglavlje funkcioniše, moramo da naznačimo sistemu da operacije mišem u ovoj oblasti odgovaraju operacijama mišem nad zaglavljem. Ovo se može postići presretanjem Windows poruke `wm_NCHitTest` koja se često šalje Windowsu da bi se odredilo gde se miš nalazi. Kada je pozicija u klijent oblasti i iznad oznake, možemo zamisliti da se miš nalazi iznad zaglavlja i proizvesti odgovarajući rezultat:

```
procedure TForm1.HitTest (var Msg: TWmNCHitTest);
  // message wm_NCHitTest
begin
  inhereted;
  if (Msg.Result = htClient) and (Msg.YPos <
      Label1.Height + Top + GetSystemMetrics (sm_cyFrame)) then
    Msg.Result := htCaption;
end;
```

API funkcija `GetSystemMetrics` se koristi u ovom listingu da bi se od operativnog sistema dobila veličina različitih vizuelnih elemenata. Važno je načiniti ovaj zahtev svaki put (i ne keširati rezultat) jer korisnici mogu da prilagode većinu ovih elemenata koristeći karticu Appearance u opcijama Desktopa (u Control Panelu) i mogu da prilagode druge Windows karakteristike. Mala kontrola sadrži poziv metoda `Close` u obradi događaja `OnClick`. Ova kontrola ostaje na svojoj poziciji čak i kada se promeni veličina prozora upotrebom vrednosti `[akTop, akRight]` svojstva `Anchors`. Formular takođe ima ograničenje veličine tako da ga korisnik ne može učiniti premalim, što je opisano u odeljku "Ograničavanje formulara" kasnije u ovom poglavlju.

Skaliranje formulara

Kada kreirate formular koji sadrži mnogo komponenata, uobičajeno je da učinite da se formularu ne može menjati veličina, da biste izbegli da neke komponente izađu iz vidljivih delova formulara. To nije veliki problem jer Delphi automatski dodaje klizače formularu, tako da do svake kontrole možete lako doći. (Skrolovanje formulara je jedna od tema u Poglavlju 7.)

Imajte na umu problem do kojeg može doći prilikom kreiranja velikog formulara: ukoliko izradite formular kada imate visoku rezoluciju na ekranu, moguće je da će biti veći od veličine ekrana na sistemima korisnika. Baš šteta, a i mnogo je češće nego što očekujete. Ukoliko možete, nikada nemojte načiniti formular veći od 640 x 480 piksela.

Ukoliko morate da načinite veći formular, a upotreba klizača nije rešenje, Delphi sadrži neke zgodne karakteristike skaliranja. Postoje dve osnovne tehnike:

- Metod formulara `ScaleBy` Vam omogućava da skalirate formular i svaku od njegovih komponenata. Ovaj metod možete upotrebiti pri pokretanju, pošto odredite rezoluciju ekrana, ili kao odgovor na specifičan zahtev korisnika.
- Svojstva `PixelPerInch` i `Scaled` omogućavaju Delphiju da automatski promeni veličinu aplikacije kada se aplikacija izvršava na sistemu sa drugačijom veličinom fonta i drugačijom ekranskom rezolucijom. Naravno, možete ručno da promenite vrednosti ovih svojstava, kao što je opisano u narednom odeljku, i da prepustite sistemu da skalira formular samo kada Vi to želite.

NAPOMENA

Skaliranje formulara se proračunava na osnovu razlike u visini fonta u vreme izvršavanja i visini fonta u vreme dizajniranja. Skaliranje obezbeđuje da su kontrola za izmene i druge kontrole dovoljno velike da prikažu tekst koristeći korisničko podešavanje fonta, a da se ne odseče deo teksta. Formular se takođe skalira, što ćemo kasnije videti, ali je mnogo važnije učiniti da kontrola za izmene i druge kontrole budu vidljive. ■

U oba slučaja, da biste učinili da formular skalira svoj prozor, postarajte se da za svojstvo `AutoScroll` odaberete vrednost `False`. U suprotnom će se skalirati sadržaj formulara, ali se neće skalirati bordura.

Ručno skaliranje formulara

Svaki put kada želite da skalirate formular, uključujući i njegove komponente, možete upotrebiti metod `ScaleBy`, koji ima dva celobrojna parametra, množilac i delilac — to je razlomak. Isti metod možete primeniti i za komponentu. Na primer, sledećim iskazom

```
ScaleBy (3, 4);
```

veličina trenutno prikazanog formulara se deli sa 4, a množi sa 3, to jest, formular je smanjen na tri četvrtine prvobitne veličine. Uopšte, lakše je koristiti procente. Isti iskaz se može napisati na sledeći način:

```
ScaleBy (75, 100);
```

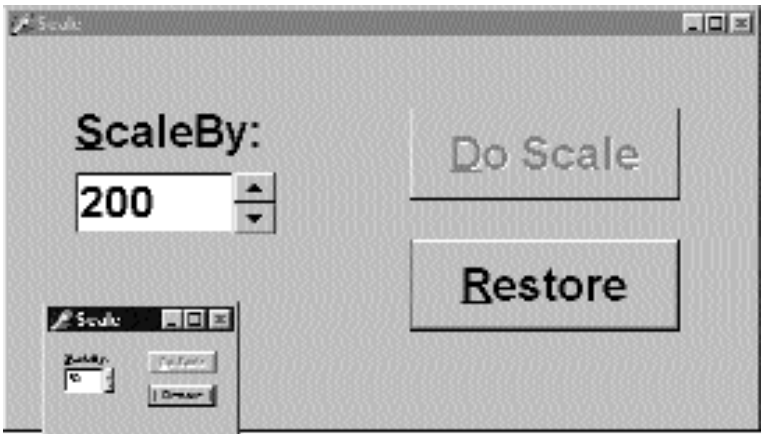
Kada skalirate formular, zadržavaju se sve proporcije, ali ukoliko idete iznad ili ispod određenog limita, tekstualni stringovi mogu malo da izgube proporcije. Ukoliko previše smanjite veličinu formulara, većina kontrola će postati neupotrebljiva ili će čak potpuno nestati. Problem je u tome što se pod Windowsom komponente mogu postaviti ili im se može promeniti veličina samo za ceo broj piksela, dok skaliranje gotovo uvek dovodi do razlomljenih brojeva. Dakle, bilo koji razlomljeni deo pozicije komponente ili njene veličine će biti odbačen.

Da biste izbegli slične probleme, trebalo bi da korisnicima omogućite samo ograničen broj operacija skaliranja ili da iznova kreirate formular pre svakog novog skaliranja da se greške zaokruživanja ne bi akumulirale.

UPOZORENJE

Ukoliko za formular primenite metod `ScaleBy`, formular neće zapravo biti skaliran. Samo će komponente unutar formulara promeniti veličinu. Kao što sam ranije pomenuo, da biste izbegli ovaj problem, potrebno je da za svojstvo `AutoSize` odaberete vrednost False. Kakva je veza između skaliranja i skrolovanja? Moja pretpostavka je da, ukoliko onemogućite skrolovanje, komponenta može da se premesti van vidljive oblasti formulara bez mnogo problema; u suprotnom, formular takođe menja veličinu. ■

Ja sam načinio jednostavan primer, primer Scale, da bih Vam pokazao kako možete ručno da skalirate formular kao odgovor na zahtev korisnika. Formular aplikacije (videti sliku 6.7) sadrži dve kontrole, oznaku, polje za izmene i kontrolu UpDown.



SLIKA 6.7 *Formular primera Scale posle skaliranja vrednostima 50 i 200*

Komponenta UpDown povezuje polje za izmene upotrebom svojstva `Associate`. Sa ovim vrednostima korisnik može uneti broj u polje ili može kliknuti jednu od dve male strelice da bi povećao ili smanjio broj u polju za izmene za fiksiranu vrednost (naznačenu svojstvom `Increment` komponente UpDown). Da biste izdvojili unetu vrednost, možete upotrebiti svojstvo `Text` polja za izmene ili svojstvo `Position` kontrole UpDown. Takođe, možete sprečiti greške prilikom unosa korisnika određivanjem svojstava Min i Max kontrole UpDown, kao što sam ja učinio u ovom primeru:

```
object UpDown1: TUpDown
  Associate = Edit1
  Min = 30
  Max = 300
  Increment = 10
  Position = 100
  Wrap = False
end
```

Kada kliknete kontrolu ScaleButton, trenutno unesena vrednost se koristi za određivanje procenta skaliranja formulara:

```
procedure TForm1.ScaledButtonClick (Sender: TObject);
begin
  AmountScaled := UpDown1.Position;
  ScaledBy (AmountScaled, 100);
  UpDown1.Height := Edit1.Height;
  ScaleButton.Enabled := False;
  RestoreButton.Enabled := True;
end;
```

Ovaj metod čuva trenutno unesenu vrednost u privatnom polju formulara `AmountScaled` i aktivira kontrolu Restore, onemogućavajući kontrolu koju ste kliknuli. Kasnije, kada korisnik klikne kontrolu Restore, obaviće se suprotno skaliranje, pozivom `ScaleBy (100, AmountScaled)`. U oba slučaja sam dodao liniju koda za određivanje svojstva Height komponente UpDown na istu vrednost svojstva `Height` polja za izmene koje je komponenti pridruženo. Na ovaj način se sprečavaju male razlike između ove dve komponente.

NAPOMENA

Ukoliko želite da pravilno skalirate tekst formulara, uključujući zaglavlja komponenata, elemente u listama i tako dalje, potrebno je da koristite samo TrueType fontove. Sistemski font (MS Sans Serif) se ne skalira pravilno. Font je veoma bitan jer veličina mnogih komponenata zavisi od visine teksta njihovih zaglavlja i, ukoliko se zaglavlje ne skalira pravilno, komponenta možda neće dobro funkcionisati. Zbog toga sam u primeru Scale koristio font Arial. ■

Automatsko skaliranje formulara

Umesto da se mučite upotrebom metoda `ScaleBy`, možete zatražiti od Delphija da posao obavi za Vas. Kada se Delphi pokrene, on od sistema zahteva konfiguraciju displeja, a vrednost čuva u svojstvu `PixelPerInch` obejkta `Screen`, specijalnog globalnog VCL objekta koji je dostupan u svakoj aplikaciji.

`PixelsPerInch` zvuči kao nešto što ima veze sa rezolucijom ekrana, ali na nesreću nema nikakve veze. Ukoliko promenite ekransku rezoluciju iz 640 x480 u 800 x 600, u 1024 x 768 ili čak u 1600 x 1280, videćete da Windows daje istu vrednost `PixelsPerInch` u svim slučajevima, izuzev ukoliko ne promenite sistemski font. Ono na šta se zaista `PixelsPerInch` odnosi je ekranska rezolucija u pikselima za koju je sistemski font dizajniran. Kada krajnji korisnik promeni veličinu sistemskog fonta, obično da bi meniji i drugi tekst bili čitljiviji, on će očekivati da sve aplikacije prate ove izmene. Aplikacija koja ne sledi vrednosti za radnu površinu korisnika neće dobro izgledati, a u ekstremnim slučajevima neće biti korisna onima kojima je potreban veoma veliki font i šema boja velikog kontrasta.

Najčešća vrednost `PixelsPerInch` je 96 (mali font) i 120 (veliki font), ali su moguće i ostale vrednosti. Windows 98 čak omogućava korisnicima da za sistemski font odrede proizvoljnu veličinu. U vreme dizajniranja vrednost ekrana `PixelsPerInch`, a to je svojstvo samo za čitanje, se kopira za svaki formular aplikacije. Delphi zatim koristi vrednost `PixelsPerInch` ukoliko je određena vrednost `True` za svojstvo `Scaled`, da bi promenio veličinu formulara kada se aplikacija pokrene.

Kao što sam već pomenuo, i automatsko skaliranje i skaliranje metodom `ScaleBy` menjaju komponente na osnovu veličine fonta. Veličina svake kontrole, zapravo, zavisi od fonta koji se koristi. Kod automatskog skaliranja, vrednost svojstva formulara `PixelsPerInch` (vrednost u vreme dizajniranja) se poredi sa trenutnom sistemskom vrednošću (naznačenom odgovarajućim svojstvom objekta `Screen`), a rezultat se koristi za promenu fonta komponenata formulara. Zapravo, da bih unapredio tačnost ovog koda, konačna visina teksta se poredi sa visinom teksta u vreme dizajniranja, a njegova veličina se prilagođava ukoliko ne odgovara.

Zahvaljujući automatskoj podršci u Delphiju, ista aplikacija koja se izvršava na sistemu sa drugačijom veličinom fonta automatski se sama skalira, bez nekog specijalnog koda. Kontrole za izmene aplikacije će biti pravilne veličine da bi prikazale svoj tekst u veličini koju zahteva korisnik, a formular će biti pravilne veličine da bi obuhvatio svoje kontrole. Mada automatsko skaliranje ima probleme u nekim specijalnim slučajevima, ukoliko sledite naredna pravila, dobićete dobre rezultate:

- Za svojstvo `Scaled` formulara odaberite vrednost `True`. (Ovo je unapred određeno.)
- Koristite samo TrueType fontove.
- Koristite Windows male fontove (96 dpi) na kompjuteru koji koristite za razvoj formulara.
- Za svojstvo `AutoScroll` odaberite vrednost `False` ukoliko želite da skalirate formular, a ne samo kontrole unutar formulara. (Unapred određena vrednost za AutoScroll je True, te nemojte zaboraviti ovaj korak.)
- Odredite poziciju formulara blizu gornjeg levog ugla ili u centru ekrana (koristeći vrednost `poScreenCenter`) da biste izbegli da formular bude van ekrana. Pozicije formulara ćemo razmatrati u narednom odeljku.

Određivanje pozicije i veličine formulara

Pored svojstva `PixelsPerInch` postoje i druga svojstva koja možete da odredite u vreme izvršavanja da biste kontrolisali izgled formulara. Svojstvo `Position` označava početnu poziciju formulara na ekranu kada je formular prvi put kreiran. Unapred određena vrednost `poDesigned` označava da će se formular prikazati tamo gde ste ga dizajnirali i koristiće se svojstva pozicioniranja i veličine formulara. Neke od ostalih vrednosti (`poDefault`, `poDefaultPosOnly` i `poDefaultSizeOnly`) zavise od karakteristika sistema: upotrebom specijalne zastavice Windows može da pozicionira i/ili odredi veličinu novog prozora koristeći kaskadno uređenje. Konačno, upotrebom vrednosti `poScreenCenter` formular se prikazuje u centru ekrana, a veličina je određena u vreme dizajniranja.

UPOZORENJE

Unapred određene pozicije se zanemaruju kada formular ima stil bordure okvira za dijalog. ■

Drugi parametar koji utiče na početnu veličinu i poziciju prozora je njegov status (state). Svojstvo `WindowState` možete koristiti u vreme dizajniranja da biste prilikom pokretanja prikazali umanjen prozor ili prozor preko cele površine ekrana. Ovo svojstvo, zapravo, može da ima tri

vrednosti: `wsNormal`, `wsMinimized` i `wsMaximized`. Značenje ovog svojstva je intuitivno. Ukoliko umanjujete prozor, verovatno će biti prikazan na Windows Taskbaru.

Naravno, prozor možete da umanjite ili da ga prikažete preko celog ekrana u vreme izvršavanja. Jednostavno, promena vrednosti svojstva `WindowState` u `wsMaximized` ili `wsNormal` daje očekivani efekat. Određivanje vrednosti `wsMinimized` kreira umanjen prozor koji se prikazuje preko Taskbara, a ne unutar njega. Ovo nije očekivana akcija za glavni formular, ali jeste za sekundarni formular! Jednostavno rešenje ovog problema je poziv metoda `Minimize` objekta `Application`. U klasi `TApplication` postoji i metod `Restore` koji možete koristiti kada je potrebno da restaurirate veličinu formulara, mada će korisnici ovu operaciju češće obaviti upotrebom komande Restore iz sistemskog menija.

Veličina formulara i njegova klijent oblast

U vreme dizajniranja postoje dva načina za određivanje veličine formulara: određivanjem svojstava `Width` i `Height` ili povlačenjem bordura formulara. U vreme izvršavanja, ukoliko formular ima borduru kojom se može menjati veličina, korisnik može da promeni veličinu (proizvevši događaj `OnResize`).

Ipak, ukoliko pogledate svojstva formulara u izvornom kodu ili Help, možete videti da postoje dva svojstva koja se odnose na širinu formulara i da postoje dva svojstva koja se odnose na visinu formulara. `Height` i `Width` se odnose na veličinu formulara uključujući i bordure; `ClientHeight` i `ClientWidth` se odnose na veličinu interne oblasti formulara isključujući bordure, zaglavlje, klizače (ukoliko ih ima) i liniju menija. Klijent oblast formulara je površina koju možete da upotrebite za smeštanje komponenata na formular, da proizvedete izlaz i da prihvatite unos korisnika.

Pošto možete da budete zainteresovani za određenu korisnu površinu, ponekad ima smisla podesiti klijentsku oblast umesto da podesite globalnu veličinu formulara. Ovo je direktna operacija jer, ukoliko podesite jedno od dva klijentska svojstva, odgovarajuće svojstvo formulara se menja prema promeni. Kada promenite vrednost svojstva `ClientHeight`, odmah se menja i vrednost svojstva `Height`.

SAVET

U Windowsu je, takođe, moguće kreirati izlaz i prihvatiti unos iz oblasti koja nije klijent formulara, to jest, sa bordure. Bojenje bordure i prihvatanje unosa kada kliknete borduru su složene operacije. Ukoliko ste zainteresovani, pogledajte Help fajl sa opisom Windows poruka kao što su poruke `wm_NCPaint`, `wm_NCCalcSize` i `wm_NCHitTest` i seriju ne-klijent poruka koje imaju veze sa mišem, kao što je `wm_NCLButtonDown`. Teškoća kod ovog pristupa je kombinovanje Vašeg koda sa unapred određenim ponašanjem Windowsa. Ipak, Delphi Vam omogućava da obradite ove Windows poruke niskog nivoa bez ikakvih problema; to je nešto što većina vizuelnih programskih okruženja ne dopušta. ■

Ograničavanje formulara

Kada za formular odaberete borduru kojom može da se menja veličina formulara, korisnici mogu da promene veličinu formulara kako žele, a takođe mogu da odrede veličinu formulara tako da prekriva ceo ekran. Windows Vas informiše o promeni veličine formulara porukom `wm_Size` koja generiše događaj `OnResize`. Događaj `OnResize` se dešava posle promene veličine formulara.

Promena veličine u okviru događaja (ukoliko je korisnik previše smanjio ili povećao formular) bi bila besmislena. Preventivan pristup je mnogo bolje rešenje ovakvog problema.

Delphi obezbeđuje specifično svojstvo za formulare, a i za sve kontrole: svojstvo `Constraints`. Jednostavno postavljanje podsvojstava svojstva `Constraints` na pravilne minimalne i maksimalne vrednosti kreira formular kome se ne može promeniti veličina van tih ograničenja. Evo primera:

```
object Form1: TForm1
  Width = 242
  Height = 162
  Constraints.MaxHeight = 300
  Constraints.MaxWidht = 300
  Constraints.MinHeight = 150
  Constraints.MinWidth = 150
end
```

Primetićete da, kada odredite svojstvo `Constraints`, vrednosti odmah proizvode efekat čak i u vreme dizajniranja, menjajući veličinu formulara ukoliko formular izlazi iz dozvoljene oblasti.

Delphi takođe koristi maksimalno ograničenje za prozor maksimalne veličine proizvodeći čudan efekat. Zbog toga bi, uopšte uzev, trebalo da onemogućite kontrolu Maximize prozora za koji je određena maksimalna veličina. Postoje slučajevi kada prozori maksimalne veličine za koje je određena maksimalna veličina imaju smisla — takvo je ponašanje Delphijevog glavnog prozora.

NAPOMENA

Svojstvo `Constraints` ima još važniju ulogu kod kontrola i za operacije dokiranja kontrola, što ćemo videti u Poglavlju 7. ■

U slučaju da je potrebno da promenite ograničenja u vreme izvršavanja, možete uzeti u obzir upotrebu dva specifična događaja, događaje `OnCanResize` i `OnConstrainedResize`. Prvi od njih se može upotrebiti za onemogućavanje promene veličine formulara ili kontrole u datim situacijama.

Kreiranje formulara

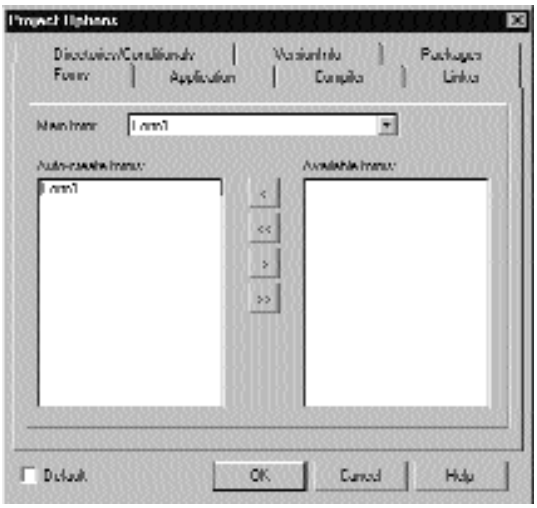
Do sada smo ignorisali temu kreiranja formulara. Znamo da kada se kreira formular, dobijamo događaj `OnCreate` i da možemo da promenimo ili testiramo neka početna svojstva formulara ili polja formulara. Iskazi odgovorni za kreiranje formulara se nalaze u ovom izvornom fajlu projekta (ili `DPR` fajlu do kojeg možete doći upotrebom komande Project sa menija):

```
begin
  Application.Initialize;
  Application.CreateForm (TForm1, Form1);
  Application.Run;
end.
```

Da biste preskočili automatsko kreiranje formulara, možete ili da izmenite ovaj kod, ili da upotrebite stranu Forms okvira za dijalog Project Options (videti sliku 6.8). U ovom okviru za dijalog možete odrediti da li formular treba automatski kreirati. Ukoliko onemogućite automatsko kreiranje, inicijalizacioni kod projekta će izgledati ovako:

```
begin
  Application.Initialize;
  Application.Run;
end.
```

Ukoliko sada pokrenete program, ništa se neće dogoditi. Program će odmah prestati sa izvršavanjem jer se neće kreirati glavni prozor. Dakle, kakav je efekat poziva metoda aplikacije `CreateForm`? Ovaj metod kreira novu instancu klase formulara koja se prosleđuje kao prvi parametar i dodeljuje je promenljivoj koja se prosleđuje kao drugi parametar.



SLIKA 6.8 *Strana Forms Delphijevog okvira za dijalog Project Options*

Još nešto se dešava iza kulisa. Kada se pozove `CreateForm`, ukoliko ne postoji glavni formular, trenutni prozor se dodeljuje svojstvu aplikacije *MainForm*. Zbog toga formular naznačen kao Main form u okviru za dijalog prikazanom na slici 6.8 odgovara prvom pozivu metoda aplikacije `CreateForm` (to jest, kada se pri pokretanju kreira nekoliko formulara).

Isto važi i za zatvaranje aplikacije. Zatvaranje glavnog formulara prekida izvršavanje aplikacije bez obzira na ostale formulare. Ukoliko ovu operaciju želite da izvršite kodom programa, jednostavno pozovite metod Close glavnog formulara, kao što smo to učinili nekoliko puta u prethodnim primerima.

SAVET

U Delphiju 5 možete (konačno) kontrolisati automatsko kreiranje sekundarnih formulara upotrebom polja za potvrdu Auto Create Forms koje se nalazi na strani Preferences okvira za dijalog Environment Options. ■

Redosled kreiranja formulara u Delphiju

Bez obzira na to da li je u pitanju automatsko ili ručno kreiranje formulara, kada kreirate formular, postoji dosta događaja koje možete presresti. Događaji kreiranja formulara se iniciraju u sledećem redosledu:

1. `OnCreate` označava da se formular kreira.
2. `OnShow` označava da se formular prikazuje. Osim za glavne formulare, ovaj događaj se dešava kada za svojstvo `Visible` formulara odredite vrednost `True` ili pozovete metode `Show` ili `ShowModal`. Ovaj događaj se poziva ukoliko je formular sakriven pa se zatim ponovo prikaže.
3. `OnActivate` označava da je formular postao aktivan formular aplikacije. Ovaj događaj se inicira svaki put kada pređete sa nekog drugog formulara aplikacije na aktuelni formular, kao što smo videli u odeljku "Aktiviranje aplikacija i formulara".
4. Ostali događaji, uključujući događaje `OnResize` i `OnPaint`, označavaju operacije koje se uvek izvršavaju prilikom pokretanja, a koje se zatim ponavljaju mnogo puta.

Kao što možete da vidite iz prethodnog spiska, svaki događaj ima određenu ulogu koja je nezavisna od inicijalizacije formulara, izuzev događaja `OnCreate` koji se sasvim sigurno poziva samo jednom prilikom kreiranja formulara.

Ipak, postoji alternativni pristup dodavanju inicijalizacionog koda formularu: to je zaobilaženje konstruktora. To se obično obavlja na sledeći način:

```
constructor TForm1.Create (AOwner: TComponent);
begin
  inherited Create (AOwner);
  // extra initialization code
end;
```

Pre poziva metoda `Create` osnovne klase, svojstva formulara još nisu čitana, a interne komponente još uvek nisu dostupne. Zbog toga je standardni pristup da se prvo pozove konstruktor osnovne klase, a da se zatim obave operacije.

Sada je pitanje da li se te operacije izvršavaju pre ili posle iniciranja događaja `OnCreate`. Odgovor zavisi od vrednosti svojstva formulara `OldCreateOrder`, koje je predstavljeno u Delphiju 4 zbog kompatibilnosti sa ranijim verzijama Delphija. (Ovo svojstvo je deo kategorije Legacy koja je u Delphiju 5 po definiciji sakrivena.) Za sve nove projekte je unapred određeno da se sav kod konstruktora izvršava pre obrade događaja `OnCreate`. Zapravo, ovu obradu događaja ne aktivira konstruktor osnovne klase već njegov metod `AfterConstruction`, tip konstruktora koji je uveden zbog kompatibilnosti sa C++ Builderom.

NAPOMENA

Da biste proučili redosled kreiranja i potencijalne probleme, možete da pogledate program CreatOrd. Ovaj program sadrži obradu događaja `OnCreate` koji dinamički kreira kontrolu ListBox. Konstruktor formulara može ili ne može pristupiti listi u zavisnosti od vrednosti svojstva `OldCreateOrder`. ■

Praćenje formulara upotrebom objekta Screen

Do sada smo već proučili neka svojstva i događaje objekta `Application`. Ostale interesantne globalne informacije o aplikaciji su dostupne preko objekta `Screen`, čija je osnovna klasa `TScreen`. Ovaj objekat čuva informacije o sistemskom displeju (veličini ekrana i ekranskim fontovima), a čuva i informacije o skupu formulara aplikacije koja se izvršava. Na primer, Vi možete da prikažete veličinu ekrana i spisak fontova ukoliko napišete:

```
Label1.Caption IntToStr (Screen.Width) + 'x' +
  IntToStr (Screen.Height);
ListBox1.Items := Screen.Fonts;
```

`TScreen` takođe može da izvesti o broju i rezoluciji monitora kod sistema sa više monitora. Ono na šta želim da se usredsredim je lista formulara koja se čuva u svojstvu `Forms` objekta `Screen`, pri čemu je formular koji se prikazuje naznačen svojstvom `ActvieForm` i odgovarajućim događajem `OnActiveFormChange`. Primetite da su formulari na koje objekat `Screen` pokazuje formulari aplikacije, a ne sistemski formulari.

Ove karakteristike su pokazane primerom Screen koji održava listu trenutnih formulara u kontroli ListBox. Ova lista mora da se ažurira svaki put kada se kreira novi formular, kada se ukloni postojeći formular, ili kada se promeni aktivni formular programa. Da biste videli kako ovo funkcioniše, možete kreirati sekundarne formulare tako što ćete kliknuti kontrolu New:

```
procedure TMainForm.NewButtonClick(Sendet: TObject);
var
  NewForm: TSecondForm;
begin
  // create a new form, set its caption, and run it
  NewForm := TSecondForm.Create (Self);
  Inc (nForms);
  NewForm.Caption := 'Second ' + IntToStr (nForms);
  NewForm. Show;
end;
```

Jedan od ključnih delova programa je obrada događaja `OnCreate` formulara koja prvo popunjava listu, a zatim povezuje obradu sa događajem `OnActiveFormChange`:

```
procedure TMainForm.FormCreate(Sender: TObject);
begin
  FillFormsList (Self);
  // set the secondary forms counter to 0
  nForms := 0;
  // set an event handler on the screen object
  Screen.OnActiveFormChange := FillFormsList;
end;
```

Kod koji se koristi za popunjavanje liste Forms se nalazi unutar druge procedure, procedure `FillFormsList`, koja je takođe instalirana i kao obrada događaja `OnActiveFormChange` objekta `Screen`:

```
procedure TMainForm.FillFormList (Sender: TObject);
var
  I: Integer;
begin
  FormsLabel .Caption := 'Forms:' +
    IntToStr (Screen.FormCount);
  Forms ListBox.Clear;
  // write class name and form title to the list box
  for I := 0 to Screen.FormCount - 1 do
    FormsListBox.Items.Add (Screen.Forms[I] .ClassName +
      '-' + Screen.Forms[I].Caption);
  ActiveLabel.Caption 'Active Form : ' +
    Screen.ActiveForm.Caption;
end;
```

UPOZORENJE

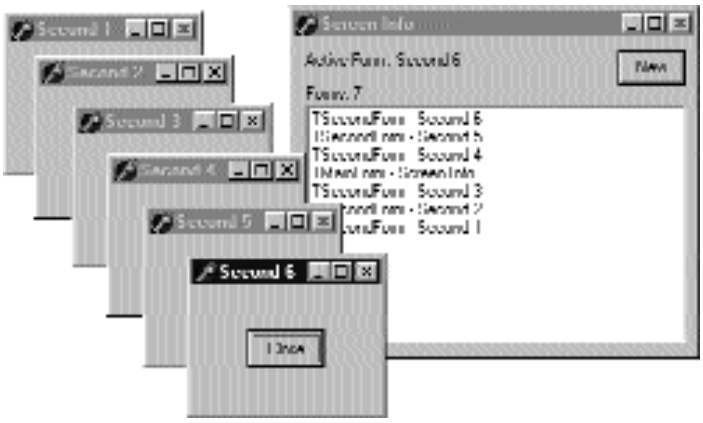
Veoma je važno da uklonite obradu događaja `OnActiveFormChange` pre nego što izađete iz aplikacije, to jest, pre nego što uklonite glavni formular. U suprotnom, kod će biti izvršen kada ne postoji ListBox i dobićete sistemsku grešku. Rešenje je u tome da obradite događaj OnClose glavnog formulara i dodelite nil za `Screen.OnActiveFormChange`. ■

Metod `FillFormsList` popunjava listu i određuje vrednost za dve oznake koje se nalaze iznad liste i kojima se prikazuje broj formulara i naziv aktivnog formulara. Kada kliknete kontrolu New, program kreira instancu sekundarnog formulara, dodeljuje mu naslov i prikazuje ga. Lista Forms se automatski ažurira zbog obrade koju smo instalirali za događaj `OnActiveFormChange`. Slika 6.9 prikazuje izlaz programa kada je kreirano nekoliko sekundarnih prozora.

SAVET

Program uvek ažurira `ActiveLabel` iznad liste da bi se prikazao trenutno aktivan formular, a to je uvek formular koji je prvi u listi. ■

Svaki od sekundarnih formulara sadrži kontrolu Close koju možete kliknuti da biste uklonili formular. Program obrađuje događaj `OnClose` dodeljujući vrednost `caFree` parametru `Action`, tako da se formular, zapravo, uklanja prilikom zatvaranja. Ovaj kod zatvara formular, ali ne ažurira listu prozora kako treba. Sistem prvo pomera fokus na neki drugi prozor liste, inicirajući događaj koji ažurira listu, i uklanja stari formular samo posle ove operacije.



SLIKA 6.9 *Izlaz primera Screen kada je kreirano nekoliko sekundarnih formulara*

Prva ideja koju sam imao za pravilno ažuriranje liste prozora je uvođenje kašnjenja slanjem korisnički definisane Windows poruke. Pošto se poslata poruka smešta u red i pošto se ne obrađuje odmah, da sam je poslao u poslednjem mogućem momentu postojanja sekundarnog formulara, glavni formular bi je dobio onda kada je drugi formular već uklonjen.

Trik je u tome što poruku treba poslati iz obrade događaja `OnDestroy` sekundarnog formulara. Da bismo ovo postigli, potrebno je da se referišemo na objekat `MainForm` dodavanjem iskaza `uses` u implementacioni deo jedinice. Ja sam poslao poruku `wm_User` koju obrađuje specifični message metod glavnog formulara kao što je to ovde pokazano:

```
public
  procedure ChildClosed (var Message: TMessage);
    message wm_User;
```

Evo koda za ovaj metod:

```
procedure TMainForm.ChildClosed (var Message: TMessage);
begin
  FillFormsList (self);
end;
```

Problem koji se ovde javlja je u tome da ukoliko zatvorite glavni prozor pre zatvaranja sekundarnih formulara, izlazi se iz glavnog formulara, ali se njegov kod više ne može izvršiti. Da biste izbegli sistemsku grešku (Access Violation Fault), potrebno je da pošaljete poruku samo ukoliko se glavni formular ne zatvara. Ali, kako da to znate? Jedan način je dodavanje zastavice klasi `TMainForm` i promena vrednosti zastavice kada se glavni formular zatvara, tako da možete testirati vrednost zastavice iz koda sekundarnog formulara.

Ovo je dobro rešenje — toliko dobro da VCL već obezbeđuje nešto slično. Postoji slabo dokumentovano svojstvo `ComponentState`. To je Pascalov skup koji sadrži (između ostalih zastavica) i zastavicu `csDestroying`, koja se postavlja kada se formular zatvara. Zbog toga možemo napisati sledeći kod:

```
procedure TSecondForm.FormDestroy (Sender: TObject);
begin
  if not (csDestroying in MainForm.ComponentState) then
    PostMessage (MainForm.Handle, wm_User, 0, 0);
end;
```

Ovim kodom lista uvek prikazuje sve formulare aplikacije. Primetite da je potrebno da onemogućite automatsko kreiranje sekundarnog formulara upotrebom strane Forms okvira za dijalog Project Options.

Kada sam razmislio, pronašao sam alternativno rešenje koje je mnogo više orijentisano ka Delphiju. Svaki put kada se komponenta ukloni, ona svom vlasniku javlja šta se dešava pozivanjem metoda `Notification` definisanom u klasi `TComponent`. Kako je glavni formular vlasnik sekundarnih formulara, što je naznačeno kodom metoda `NewButtonClick`, mi možemo zaobići ovaj metod i pojednostaviti kod. U klasi formulara jednostavno napišite:

```
protected
  procedure Notification (AComponent: TComponent;
    Operation: TOperation); override;
```

Evo koda metoda:

```
procedure TMainForm.Notification (AComponent: TComponent;
  Operation: TOperation);
begin
  inherited Notofication (AComponent, Operation);
  if Showing and (Acomponent is TForm) then
    FillFormsList;
end;
```

Kompletan kod ove verzije ćete pronaći u direktorijumu Screen2.

NAPOMENA

U slučaju da vlasnik sekundarnih formulara nije glavni formular, mogli bismo da upotrebimo metod `FreeNotification` da bi sekundarni formular obavestio glavni formular o svom uklanjanju. Metod `FreeNotification` kao parametar dobija komponentu da bi obavestio kada se komponenta uklanja. Efekat je poziv metoda `FreeNotification` koji dolazi od komponente koja nije među komponentama vlasnika. Metod `FreeNotification` koriste programeri komponenata da bi bezbedno povezali komponente različitih formulara ili modula podataka. ■

Poslednja karakteristika koju sam dodao obema verzijama programa je jednostavna. Kada kliknete element liste, odgovarajući formular se aktivira upotrebom metoda `BringToFront`:

```
procedure TMainForm.FormsListBoxClick (Sender: TObject);
begin
  Screen.Forms [FormsListBox.ItemIndex].BringToFront;
end;
```

Lepo — zapravo, gotovo da je tako. Ukoliko kliknete listu neaktivnog formulara, prvo se aktivira glavni formular, lista se ponovo generiše, tako da se može desiti da se selektuje neki drugi formular od onog koji ste očekivali. Ukoliko eksperimentišete programom, vrlo brzo ćete uvideti na šta sam mislio. Ovaj manji propust u programu je primer rizika sa kojim se suočavate kada dinamički ažurirate neke informacije i istovremeno dopustite korisniku da na njima radi.

Zatvaranje formulara

Kada zatvorite formular upotrebom metoda Close ili upotrebom uobičajenih načina (Alt+F4, sistemski meni ili kontrola Close), poziva se događaj `OnCloseQuery`. U tom slučaju možete zatražiti da korisnik potvrdi akciju, naročito ukoliko postoje nesačuvani podaci na formularu. Evo jednostavne šeme koda koji možete napisati:

```
procedure TForm1.FormCloseQuery (Sender: TObject;
  var CanClose: Boolean);
begin
  if MessageDlg ('Are you sure you want to exit?',
      mtConfirmation, [mbYes, mbNo], 0) = idNo then
    CanClose : =False;
end;
```

Ukoliko `OnCloseQuery` označava da formular i dalje treba da bude zatvoren, poziva se događaj `OnClose`. Treći korak je poziv događaja `OnDestroy`, koji je suprotan događaju `OnCreate` i koristi se dealociranje objekata koji su povezani sa formularom i oslobađanje odgovarajuće memorije.

NAPOMENA

Budimo precizniji, metod `BeforeDestruction` generiše događaj `OnDestroy` pre nego što se pozove destruktor `Destroy`, to jest, izuzev ukoliko niste odredili vrednost `True` za svojstvo `OldCreateOrder` kada Delphi koristi drugačiji redosled zatvaranja. ■

Dakle, kakva je korist od međudogađaja `OnClose`? U ovom metodu imate još jednu šansu da izbegnete zatvaranje aplikacije, ili da odredite alternativne "akcije zatvaranja". Metod, zapravo, sadrži parametar `Action` koji se prosleđuje po referenci. Parametru možete dodeliti sledeće vrednosti:

- `caNone`: Ne dozvoljava se zatvaranje formulara. Ovo odgovara određivanju vrednosti `False` parametra `CanClose` metoda `OnCloseQuery`.
- `caHide`: Formular se ne zatvara već sakriva. Ovo ima smisla ukoliko postoje i drugi formulari aplikacije; u suprotnom se prekida izvršavanje programa. Ovo je unapred određena vrednost za sekundarne formulare i to je razlog zbog kojeg sam morao da obradim događaj `OnClose` u prethodnom primeru da bih zaista zatvorio sekundarne formulare.
- `caFree`: Formular se zatvara, oslobađa se njegova memorija i prekida se izvršavanje aplikacije ukoliko je u pitanju glavni formular. Ovo je unapred određena akcija za glavni formular i akcija koju bi trebalo da korisite kada dinamički kreirate više formulara (ukoliko želite da uklonite Windows i uklonite odgovarajuće Delphi objekte kada se formular zatvori).
- `caMinimize`: Formular se ne zatvara već se samo smanjuje. Ovo je unapred određena akcija za MDI dete-formulare, što ćemo videti u Poglavlju 8.

NAPOMENA

Kada korisnik obori Windows, aktivira se događaj `OnCloseQuery` i program ga može upotrebiti za prekid procesa obaranja. U tom slučaju se događaj `OnClose` ne poziva čak i ako `OnCloseQuery` odredi vrednost True parametra `CanClose`. ■

Unos sa formulara

Do sada smo razmatrali neke specijalne mogućnosti formulara, a sada ću preći na veoma važnu temu: korisnički unos sa formulara. Ukoliko odlučite da načinite ograničenu upotrebu komponenata, možete takođe napisati složen program koji unos prihvata od miša i sa tastature. U ovom poglavlju ću samo predstaviti ovu temu. Više o grafici se može pronaći u dodatnom poglavlju "Grafika u Delphiju" koje je dostupno na adresi `www.sybex.com`.

Nadgledanje unosa sa tastature

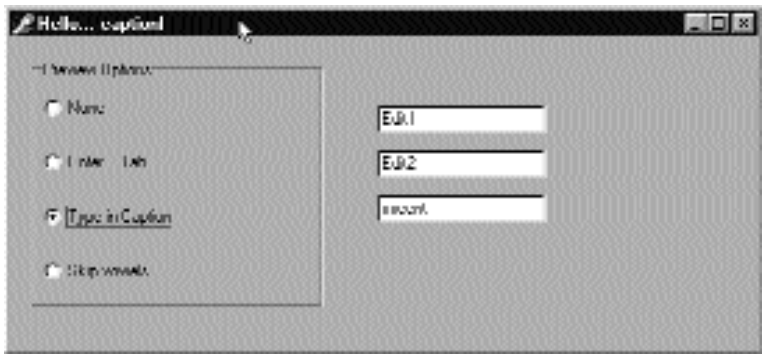
Uopšte uzev, formulari direktno ne obrađuju unos sa tastature. Ukoliko je potrebno da korisnik nešto unese, Vaš formular treba da sadrži polje za izmene ili neku od ulaznih komponenata. Ukoliko želite da obradite tastaturne prečice, možete upotrebiti one koje su povezane sa menijima (moguće koristeći iskačući meni).

U drugim slučajevima ćete možda želeti da obradite unos sa tastature na određene načine za specifične upotrebe. Ono što u tim slučajevima možete učiniti je da uključite svojstvo formulara `KeyPreview`. Zatim, čak i kad imate ulazne kontrole, uvek će se aktivirati događaj formulara `OnKeyPress` za bilo koju operaciju unosa sa tastature. Unos sa tastature će zatim dospeti do odredišne komponente, izuzev ukoliko unos ne zaustavite na formularu određivanjem vrednosti nula za karakter (ne karakter 0, već vrednost 0 iz skupa karaktera što se naznačava kao #0).

Primer koji sam načinio da bih ovo pokazao, primer KPreview, sadrži formular bez specijalnih svojstava (nema čak ni svojstva `KeyPreview`), sadrži opcione kontrole sa četiri opcije i nekoliko polja za izmene, što možete videti na slici 6.10.

Unapred je određeno da program ne čini ništa specijalno, izuzev kada se različite opcione kontrole upotrebe za omogućavanje prikaza tastera:

```
procedure TForm1.RadioPreviewClick (Sender: TObject);
begin
  KeyPreview := RadioPreview.ItemIndex <> 0;
end;
```



SLIKA 6.10 *Program KPreview Vam omogućava da unesete zaglavlje (kao i neke druge stringove)*

Sada ćemo početi da dobijamo događaje OnKeyPress i možemo učiniti jednu od tri akcije koje su zahtevane jednom od tri specijalne opcione kontrole. Akcija zavisi od vrednosti ItemIndex svojstva komponente RadioGroup. To je razlog zbog kojeg se obrada događaja zasniva na iskazu case:

```
procedure TForm1.FormKeyPress (Sender: TObject; var Key: Char);
begin
  case RadioPreview.ItemIndex of
    ...
```

U prvom slučaju, ukoliko je vrednost parametra #13, što odgovara tasteru Enter, mi ćemo onemogućiti operaciju (određivanjem nula za Key), a zatim ćemo oponašati aktiviranje tastera Tab. Postoji mnogo načina da se ovo postigne, ali je način koji sam ja odabrao prilično čest. Ja sam formularu poslao poruku CM_DialogKey, prosleđujući kod za taster Tab (VK_TAB):

```
1: // Enter = Tab
  if Key = #13 then
  begin
    Key := #0;
    Perform (CM_DialogKey, VK_TAB, 0);
end;
```

NAPOMENA

Poruka CM_DialogKey je interna nedokumentovana Delphi poruka, nešto što je zaista izvan okvira ove knjige, ali se razmatra u drugim tekstovima, uključujući i moju knjigu "Priručnik za Delphi programere" (Sybex, 1998). ■

Da bi se unelo zaglavlje formulara, program jednostavno dodaje karakter svojstvu Caption, kao što možete videti na slici 6.10. Postoje dva specijalna slučaja. Kada se pritisne taster Backspace, poslednji karakter stringa se uklanja (kopiranjem svih karaktera svojstva Caption izuzev poslednjeg karaktera). Kada se pritisne taster Enter, program prekida operaciju resetovanjem svojstva ItemIndex kontrole RadioGroup. Evo i koda:

```
2: // type in caption
begin
  if Key = #8 then // backspace: remove last char
    Caption := Copy (Caption, 1,
      Length (Caption)-1)
  else if Key = #13 then // enter: stop operation
    RadioPreview.ItemIndex := 0
  else // anything else: add character
    Caption := Caption + Key;
  Key := #0;
end;
```

Konačno, ukoliko je odabran poslednji element, kod proverava da li je karakter samoglasnik (testiranjem da li postoji u konstantnom skupu vowel). Ukoliko je to slučaj, karakter se preskače:

```
3: // skip vowels
  if Key in ['a', 'e', 'i', 'o', 'u',
    'A', 'E', 'I', 'O', 'U'] then
    Key := #0;
```

Prihvatanje ulaza pomoću miša

Kada korisnik pritisne jedan od tastera miša (kada se pokzivač miša nalazi iznad formulara ili, čak, komponente), Windows aplikaciji šalje poruke. Delphi definiše neke događaje koje možete upotrebiti da napišete kod koji će odgovoriti na ove poruke. Dva osnovna događaja su sledeća:

- `OnMouseDown` se dobija kada se pritisne jedan od tastera miša.
- `OnMouseUp` se dobija kada se otpusti jedan od tastera miša.

Još jedna od osnovnih sistemskih poruka je povezana sa kretanjem miša. To je događaj `OnMouseMove`. Mada bi trebalo da je lako razumeti značenje tri poruke — pritisnuti, otpustiti i pomeriti — pitanje koje se može javiti je kakve veze one imaju sa događajem `OnClick` koji smo do sada tako često koristili.

Mi smo koristili događaj `OnClick` za komponente, ali je ovaj događaj dostupan i za formular. Njegovo opšte značenje je da je pritisnut i otpušten taster miša nad istim prozorom ili komponentom. Ipak, između ove dve akcije kursor se može pomeriti van oblasti prozora ili komponente dok je pritisnut levi taster miša. Ukoliko pritisnete levi taster miša na određenoj poziciji, a zatim pomerite pokazivač miša i otpustite taster, tada se ne deštava ništa. U tom slučaju prozor dobija samo poruku o pritisku tastera miša, neke poruke o kretanju i poruku o otpuštanju tastera miša. Druga razlika je da se događaj Click odnosi samo na levi taster miša.

Tasteri miša

Većina tipova miševa koji su povezani sa Windows PC-jima ima dva tastera, a poneki imaju i tri. Obično te tastere nazivamo levi taster miša, koji se najviše koristi; desni taster miša i srednji taster miša:

- Levi taster miša je glavni taster miša. Koristi se za selektovanje elemenata na ekranu, za odabiranje komandi menija, koristi se da bi se kliknula kontrola, za selektovanje i pomeranje elemenata (prevlačenje — dragging), za selektovanje i aktiviranje (kada dva puta kliknete) i tako dalje.
- Desni taster miša se koristi za iskačuće menije. Mnoge aplikacije su u prošlosti koristile ovakav pristup, ali je Windows 95 ove menije načinio standardnim efektom kada kliknete desnim tasterom miša.
- Srednji taster miša se retko koristi jer većina korisnika ne poseduje miš sa tri tastera ili nema odgovarajući softver. Neki CAD programi koriste srednji taster miša. Ukoliko želite da podržite ovaj taster, to bi trebalo da bude podržano kao opcija (ili bi Vašim mušterijama trebalo da besplatno obezbedite miša sa tri tastera i odgovarajući drajver).

Imajte na umu da korisnici mogu prilagoditi tastere miša prema svojim potrebama, menjajući uloge levog i desnog tastera i pretvarajući jedan klik srednjim tasterom u dvostruki klik. Kada se referišete na događaje koji imaju veze sa tasterom miša u Vašem kodu, ono što je važno nije fizički taster već njegovo značenje.

NAPOMENA

Pored tradicionalnih miševa sa tri tastera, postoje novi miševi sa točkićem umesto srednjeg tastera. Korisnici točkić koriste za skrolovanje (što dovodi do događaja `OnMouseWheel`), ali ga takođe mogu i pritisnuti (čime se generiše događaj `OnMouseWheelDown` ili događaj `OnMouse WheelUp`). Poruke Up i Down su slične porukama tastera miša, dok događaj `OnMouseWheel` ogovara operacijama skrolovanja. Događaji točkića miša se automatski konvertuju u događaje skrolovanja. ■

Upotreba Windowsa bez upotrebe miša

Korisnicima bi trebalo omogućiti da koriste bilo koju Windows aplikaciju bez upotrebe miša. To nije opcija; to je pravilo Windows programiranja. Naravno, aplikaciju je možda lakše koristiti upotrebom miša, ali to nikada ne bi trebalo da bude obavezno. U stvari, postoje korisnici koji iz raznih razloga nemaju miša povezanog sa kompjuterom, kao što su, recimo, putnici koji imaju prenosne računare i nemaju dovoljno prostora, radnici u industrijskom okruženju ili bankarski činovnici koji oko sebe imaju mnogo perifernih uređaja.

Postoji još jedan razlog, koji je već pomenut u ovom poglavlju a odnosi se na meni, zbog kojeg treba podržati upotrebu tastature: Upotreba miša je dobra, ali usporava rad. Ukoliko ste dobar daktilograf, nećete želeti da koristite miša za prevlačenje reči teksta; upotrebićete tastaturne prečice za kopiranje i premeštanje tako da ne morate da pomerate ruke sa tastature.

Zbog svih ovih razloga trebalo bi uvek da odredite pravilan redosled prelaženja sa komponente na komponentu formulara, a trebalo bi da dodate kontrole i elemente menija koji se mogu odabrati sa tastature upotrebom tastaturnih prečica menija. Izuzetak od ovog pravila su možda grafički programi. Ipak, imajte na umu da možete koristiti program kakav je Microsoft Paint bez upotrebe miša — mada ja to ne preporučujem.

Parametri događaja miša

Budući da nameravam da načinim grafički program, ja ću obratiti pažnju samo na upotrebu miša. Prvi događaj koji je potrebno da uzmemo u obzir za prvu minimalnu verziju programa MouseOne je događaj `OnMouseDown`. Odgovarajući metod ima veliki broj parametara, kao što se može videti u narednoj deklaraciji:

```
procedure TShapesForm.FormMouseDown (
  Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
```

Pored uobičajenog parametra Sender postoje još četiri parametra:

- Parametar `Button` označava koji od tri tastera miša je pritisnut. Moguće vrednosti su `mbRight`, `mbLeft` i `mbCenter`. Ovo su ekskluzivne vrednosti jer je svrha ovog parametra određivanje tastera miša koji generiše poruku.
- Parametar `Shift` označava koji odgovarajući tasteri, koji su povezani sa mišem, su pritisnuti kada se događaj desi. Ovi tasteri mogu biti Alt, Ctrl i Shift, kao i sami tasteri miša. Ovaj prametar je tipa skup jer nekoliko tastera (i tastera miša) može istovremeno biti pritisnuto. To znači da morate da testirate uslov koristeći izraz in, a ne da testirate jednakost.

- Parametri `X` i `Y` označavaju koordinate pozicije pokazivača miša, u koordinatama *klijent oblasti* trenutnog prozora (formulara ili kontrole). Koordinatni početak `x`- i `y`-ose se nalazi u gornjem levom uglu klijent oblasti prozora koji prima događaj (ponovimo još jednom, to može biti formular ili kontrola).

Koristeći ove informacije veoma je jednostavno nacrtati mali krug na poziciji gde se odigrao događaj pritiska levog tastera miša:

```
procedure TForm1.FormMouseDown (
  Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  if Button = mbLeft then
    CanVas.Ellipse (X-10, Y-10, X+10, Y+10);
end;
```

NAPOMENA

Da bismo crtali po formularu, koristimo specijalno svojstvo `Canvas`. Objekat `TCanvas` ima dve odvojene karakteristike: čuva kolekciju alata za crtanje (kao što su recimo olovka, četkica i font) i sadrži brojne metode crtanja koji koriste trenutne alate. Ova vrsta direktnog crtanja ovog primera nije korektna jer ekranske slike nisu konstantne: pomeranje nekog prozora preko trenutnog prozora će obrisati izlaz. Naredni primer pokazuje Windowsov pristup "čuvaj-i-crtaj". ■

Prevlačenje i crtanje upotrebom miša

Da bih demonstrirao nekoliko tehnika mišem koje smo do sada razmatrali, napisao sam jednostavan primer zasnovan na formularu bez komponenata koji sam nazvao MouseOne. Prva karakteristika programa je prikazivanje zaglavlja formulara na trenutnoj poziciji miša:

```
procedure TMouseForm.FormMouseMove (Sender: TObject;
  Shift: TShiftState; X, Y: Integer);
begin
  // display the position of the mouse in the caption
  Caption := Format ('Mouse in x=%d, y=%d', [X, Y]);
end;
```

Možete da upotrebite ovu jednostavnu karakteristiku programa da biste bolje razumeli kako miš funkcioniše. Načinite sledeći test: poruke miša su uvek upućene prozoru koji se nalazi ispod miša. Jedini izuzetak je operacija "hvatanja" miša koju ću razmatrati u istom primeru.

Pored prikazivanja pozicije u naslovu prozora, primer MouseOne može da prati kretanje miša iscrtavanjem malih piksela na formularu ukoliko korisnik drži pritisnut taster Shift. (Ponoviću još jednom, ovaj kod direktnog crtanja daje izlaz koji nije trajan.)

```
procedure TMouseForm.FormMouseMove (Sender: TObject;
  Shift: TShiftState; X, Y: Integer);
begin
  // display the position of the mouse in the caption
  Caption := Format ('Mouse in x=%d, y=%d', [X, Y]);
  if ssShift in Shift then
    // mark points in yellow
    CanVas.Pixels [X,Y] := clYellow;
end;
```

Prava karakteristika ovog primera je podrška direktnom prevlačenju miša. Nasuprot onome što možda mislite, Windows nema podršku za prevlačenje, koja je implementirana u VCL-u preko događaja i operacija niskog nivoa. (Primer prevlačenja sa jedne kontrole na drugu je bio razmatran u prethodnom poglavlju.) U VCL-u formulari ne mogu da započnu operaciju prevlačenja, te smo mi u tom slučaju obavezni da koristimo pristup na niskom nivou. Cilj ovog primera je crtanje pravougaonika počevši od inicijalne pozicije operacije prevlačenja do krajnje pozicije operacije prevlačenja dajući korisnicima vizuelno rešenje operacije koja se odvija.

Ideja koja stoji iza prevlačenja je prilično jednostavna. Program dobija niz poruka o pritisnutom tasteru, pomeranju miša i otpuštanju tastera. Kada je taster pritisnut, počinje prevlačenje, mada se stvarna akcija dešava samo kada korisnik pomeri miša (ne otpuštajući taster miša) i kada se prevlačenje završi (kada pristigne poruka da je taster otpušten).

Problem ovog osnovnog pristupa je u tome što nije pouzdan. Prozor obično dobija događaj miša samo kada se miš nalazi iznad njegove klijent oblasti; dakle, ukoliko korisnik pritisne taster miša, pomeri pokazivač miša iznad nekog drugog prozora, a zatim otpusti taster miša, drugi prozor će dobiti poruku o otpuštanju miša.

Postoje dva rešenja ovog problema. Jedan (koji se retko koristi) je "odsecanje" miša. Koristeći Windows API funkciju (drugim rečima `ClipCursor`) možete obezbediti da se pokazivač miša ne može pomeriti izvan određene oblasti ekrana. Kada pokušate da ga pomerite van određene oblasti, on će se odbiti od nevidljive barijere. Kada prozor "zatvori" pokazivač miša, sav naredni ulaz miša se šalje tom prozoru. Ovo je pristup koji ćemo koristiti za primer MouseOne.

Kod primera je baziran na osnovu tri metoda: `FormMouseDown`, `FormMouseMove` i `FormMouseUp`. Kada se pritisne levi taster miša iznad formulara, započinje proces, određivanjem vrednosti Boolean polja formulara `fDragging` (koje u ostala dva metoda označava da se odvija prevlačenje). Metod takođe koristi promenljivu `TRect` da bi pratio početnu i trenutnu poziciju prevlačenja. Evo koda:

```
procedure TMouseForm.FormMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  if Button = mbLeft then
  begin
    fDragging := True;
    SetCapture (Handle);
    fRect.Left := X;
    fRect.Top := Y;
    fRect.BottomRight := fRect.TopLeft;
    CanVas. DrawFocusRect (fRect);
  end;
end;
```

Važna akcija ovog metoda je poziv API funkcije `SetCapture`. Sada čak i da korisnik pomeri pokazivač miša izvan klijent oblasti, formular još uvek dobija sve poruke koje se tiču miša. U to se i sami možete uveriti pomeranjem pokazivača miša ka gornjem levom uglu ekrana; program će u zaglavlju prikazivati negativne koordinate.

Kada je prevlačenje aktivno i kada korisnik pomera miša, program iscrtava tačkasti pravougaonik koji odgovara trenutnoj poziciji. Zapravo, program dva puta poziva metod DrawFocusRect. Prvi

put kada se ovaj metod pozove, on uklanja trenutnu sliku zahvaljujući činjenici da dva uzastopna poziva metoda DrawFocusRect jednostavno restauriraju početnu situaciju. Posle ažuriranja pozicije pravougaonuka program drugi put poziva metod:

```
procedure TMouseForm.FormMouseMove(Sender: TObject;
  Shift: TShiftState; X, Y: Integer);
begin
  // display the position of the mouse in the caption
  Caption := Format ('Mouse in x=%d, y=%d' , [X, Y]);
  if fDragging then
  begin
    // remove and redraw the dragging rectangle
    CanVas .DrawFocusRect (fRect);
    fRect.Right := X;
    fRect.Bottom := Y;
    CanVas.DrawFocusRect (fRect);
  end
  else
    if ssShift in Shift then
      // mark points in yellow
      CanVas.Pixels [X, Y] := clYellow;
end;
```

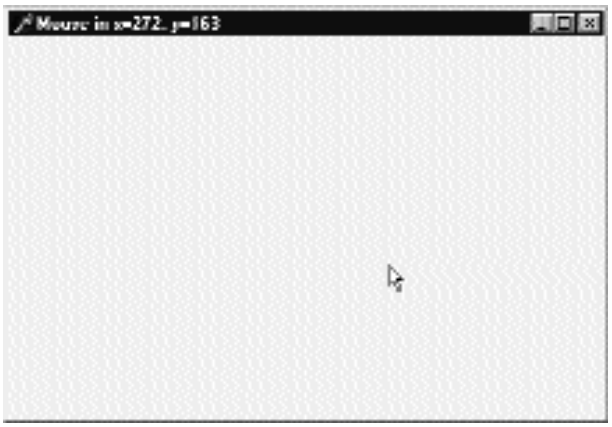
Kada se otpusti taster miša, program prekida operaciju prevlačenja pozivanjem API funkcije ReleaseCapture i određivanjem vrednosti False za polje fDragging:

```
procedure TMouseForm.FormMouseUp(Sender: TQbject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  if fDragging then
  begin
    ReleaseCapture;
    fDragging := False;
    Invalidate;
  end;
end;
```

Poslednji poziv, poziv Invalidate, pokreće operaciju iscrtavanja i izvršava sledeću obradu događaja OnPaint:

```
procedure TMouseForm.FormPaint(Sender: TObject);
begin
  CanVas.Rectangle (fRect.Left, fRect.Top,
    fRect.Right, fRect.Bottom);
end;
```

Na ovaj način izlaz programa postaje trajan, čak i ako ga sakrijete iza nekog drugog formulara. Slika 6.11 prikazuje prethodnu verziju pravougaonika i akciju operacije prevlačenja.



SLIKA 6.11 *Primer MouseOne koristi tačkastu liniju da bi prilikom operacije prevlačenja naznačio konačnu oblast pravougaonika*

Crtanje u Windowsu

Zbog čega nam je potreban događaj `OnPaint` da bismo proizveli korektan izlaz i zbog čega ne možemo direktno da iscrtamo sliku formulara? Odgovor zavisi od unapred određenog ponašanja Windowsa. Kada crtate u prozoru, Windows ne čuva rezultujuću sliku. Kada je prozor prekriven, njegov sadržaj se obično gubi.

Razlog za ovakvo ponašanje je prilično jednostavan: da bi se sačuvala memorija. Windows pretpostavlja da je na duge staze "jeftinije" ponovo iscrtavati ekran upotrebom koda nego upotrebiti deo memorije za čuvanje izgleda prozora. To je klasičan izbor između memorije i CPU ciklusa. Bitmapa u boji veličine 300 x 400 u 256 boja zahteva oko 120 KB. Povećanjem broja boja ili broja piksela Vi lako možete da imate bitmape preko celog ekrana, za koje je potrebno oko 1 MB, a koje mogu zahtevati do 4 MB memorije u rezoluciji 1280 x 1024 u 16 miliona boja. Da je čuvanje bitmape bilo rešenje, izvršavanje pola tuceta jednostavnih aplikacija bi zahtevalo najmanje 8 MB memorije, ako ne i čitavih 16 MB, samo za pamćenje njihovog izlaza.

U opštem slučaju kada želite da imate stalan izlaz Vaših aplikacija, postoje dve tehnike koje možete upotrebiti. Opšte rešenje je da sačuvate dovoljno podataka o izlazu da biste mogli da ga reprodukujete kada sistem pošalje zahtev za crtanje. Alternativan pristup je čuvanje izlaza formulara u obliku bitmape prilikom njegovog prikazivanja, tako što se postavi komponenta Image preko formulara i što se crta preko sadržaja slike.

Prva tehnika, crtanje, je uobičajen pristup za rukovanje izlazom u Windowsu, ako izuzmemo specifične grafički orijentisane programe koji formular u celosti čuvaju kao bitmapu. Pristup koji se koristi za implementiranje iscrtavanja ima veoma opisno ime: *čuvaj* i *crtaj*. Zapravo, kada korisnik pritisne taster miša ili izvrši bilo koju drugu operaciju, mi moramo da sačuvamo poziciju i druge elemente; zatim, u metodu crtanja koristimo ovu informaciju da bismo zapravo nacrtali odgovarajuću sliku.

Ideja ovog pristupa je da omogućimo aplikaciji da ponovo iscrta celu površinu pod bilo kojim okolnostima. Ukoliko obezbedimo metod za ponovno crtanje sadržaja formulara, i ukoliko se ovaj metod automatski poziva kada je deo prozora bio sakriven i kada ga je potrebno ponovo iscrtati, mi ćemo ponovo moći da pravilno kreiramo izlaz.

Pošto ovakav pristup zahteva dva koraka, mi moramo biti u mogućnosti da ove dve operacije izvršimo jednu za drugom, tražeći od sistema da ponovo iscrta prozor — a da ne čekamo da zatraži ponovno iscrtavanje. Možete upotrebiti nekoliko metoda da biste pozvali ponovno iscrtavanje: `Invalidate`, `Update`, `Repaint` i `Refresh`. Prva dva metoda odgovaraju Windows API funkcijama, dok su druga dva predstavljena u Delphiju.

- Metod `Invalidate` informiše Windows da je celu površinu formulara potrebno ponovo iscrtati. Najvažnija stvar je da metod `Invalidate` ne zahteva momentalno operaciju iscrtavanja. Windows jednostavno čuva zahtev, a na njega će odgovoriti tek kada se u potpunosti završi izvršavanje trenutne procedure, i onog trenutka kada više u sistemu nema događaja koji čekaju na izvršavanje. Windows namerno odlaže opraciju iscrtavanja jer je to jedna od operacija za koju je potrebno dosta vremena. Ponekad je uz ovo odlaganje moguće iscrtati formular tek pošto se dogode brojne izmene, izbegavajući brojne uzastopne pozive (sporog) metoda iscrtavanja.
- Metod Update zahteva od Windowsa da ažurira sadržaj formulara, momentalno ga iscrtavajući. Ipak, imajte na umu da će se ova operacija dogoditi samo ukoliko postoji promenjena oblast (invalid area). Ovo se događa ukoliko je upravo pozvan metod `Invalidate`, ili kao rezultat operacije korisnika. Ukoliko nema nepromenjene oblasti, poziv metoda `Update` nema nikakav efekat. Zbog toga obično možete videti poziv metoda `Update` odmah posle poziva metoda `Invalidate`. To je ono što je učinjeno pomoću druga dva Delphi metoda: `Repaint` i `Refresh`.
- Metod `Repaint` poziva metod `Invalidate` i metod `Update` jedan za drugim. Kao rezultat ovaj metod momentalno aktivira događaj `OnPaint`. Efekat je identičan za formular; za komponente se može malo razlikovati.

Kada je potrebno da od formulara zatražite operaciju ponovnog iscrtavanja, obično bi trebalo da pozovete `Invalidate` što je u skladu sa standardnim Windows pristupom. Ovo je naročito važno kada ovu operaciju često zahtevate, jer je Windowsu potrebno dosta vremena da ažurira ekran, te se zahtevi za ponovnim iscrtavanjem mogu akumulirati u jednu jednostavnu akciju ponovnog iscrtavanja. Poruka `wm_Paint` je u Windowsu poruka niskog prioriteta. Budimo precizniji, ukoliko zahtev za ponovnim iscrtavanjem čeka na red, ali ukoliko i ostale poruke čekaju na red, druge poruke se obrađuju pre nego što sistem zaista izvrši akciju iscrtavanja.

S druge strane, ukoliko pozovete nekoliko puta metod `Repaint`, ekran se mora ponovo iscrtati svaki put pre nego što Windows može da obradi ostale poruke, a kako je za operaciju iscrtavanja potrebno mnogo računanja, to Vaše aplikacije čini sporijim. Ponekad želite da aplikacija iscrta površinu što je brže moguće. U ovim manje čestim situacijama poziv metoda Repaint je pravo rešenje.

NAPOMENA

Još jedna stvar koju je važno uzeti u razmatranje je da tokom operacije iscrtavanja Windows ponovo iscrtava samo takozvani region ažuriranja (update region) da bi se ubrzalo izvršavanje operacije. Zbog toga, ukoliko izmenite samo deo prozora, samo taj deo će biti ponovo iscrtan. Da biste to postigli, možete da pozovete funkcije `InvalidateRect` i `InvalidateRegion`. Zapravo, ove funkcije su mač sa dve oštrice. Ovo je veoma moćna tehnika koja može povećati brzinu izvršavanja i smanjiti treptanje koje prouzrokuju česte operacije iscrtavanja. S druge strane, može dovesti do nepravilnog izlaza. Tipičan problem se javlja kada su samo delovi oblasti promenjeni korisničkim operacijama dok ostali ostaju na mestu čak i kada sistem izvrši izvorni kod koji bi trebalo da ih ažurira. Zapravo, ukoliko operacija iscrtavanja izađe iz regiona ažuriranja, sistem je ignoriše kao da je van vidljive oblasti prozora. ■

Šta je sledeće?

U ovom poglavlju smo upoznali neka važna svojstva formulara. Sada znamo kako da odredimo veličinu i poziciju formulara, kako da mu promenimo veličinu, kako da preuzmemo ulaz miša i kako da crtamo preko formulara. Takođe smo detaljno razmatrali dva globalna objekta, objekte `Application` i `Screen` i načinili smo aplikacije sa više formulara. U Poglavlju 8 ćemo proširiti ovo znanje na okvire za dijalog.

Ostala poglavlja ove knjige će se baviti temama koje su vezane za formulare. Bonus poglavlje koje možete naći na adresi `www.sybex.com` se detaljnije bavi grafičkim izlazom formulara; Poglavlje 7 se bavi upotrebom paleta alata, statusnih linija i skrolovanjem formulara; Poglavlje 8 — izradom okvira za dijalog, formulara sa više strana i MDI aplikacija. Kao što možete videti iz ovog spiska, formulari imaju centralnu ulogu u Delphi programiranju, a mi još treba da se upoznamo sa brojnim temama koje su vezane za formulare.

POGLAVLJE

7

Izrada korisničkog interfejsa

Jedna od jedinstvenih karakteristika mnogih Windows aplikacija je postojanje palete alata pri vrhu prozora i statusne linije u dnu prozora. Paleta alata obično sadrži brojne male kontrole koje korisnik može kliknuti da bi zadao komande, ili kojima može uključiti ili isključiti opcije. Paleta alata može da sadrži combo polja, polja za izmene ili neke druge kontrole. Sadašnja generacija paleta alata velikih aplikacija se obično može pomeriti na levu ili desnu stranu prozora, a čak se može i sakriti ili pretvoriti u okvir sa alatima, mali pokretni prozor sa nizom kontrola.

Složenije aplikacije često sadrže više paleta alata koje korisnik može da prilagodi. U Delphiju možete da koristite ili komponentu ControlBar ili Win32 kontrolu CoolBar, koju je prvobitno predstavio Microsoft Internet Explorer.

Paletama alata se bavi samo prvi deo ovog poglavlja koje, takođe, objašnjava podršku dokiranju, koje je predstavljeno u Delphiju 4, i prikazuje primere deljenja formulara, dinamičke promene veličine kontrola i skrolovanje sadržaja formulara. Ove teme nisu naročito složene, ali vredi da se kratko upoznate sa njihovim ključnim konceptima.

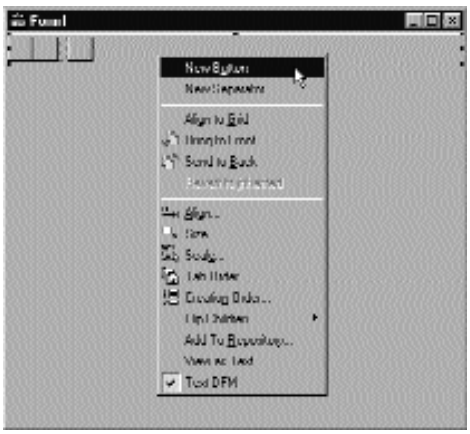
Kontrola Toolbar

U prvim verzijama Delphija palete alata je trebalo kreirati upotrebom panela i speed kontrola, što će kratko biti objašnjeno u delu "Izrada paleta alata upotrebom panela" kasnije u ovom poglavlju. Počevši od verzije 3, Delphi je predstavio specifičnu komponentu Toolbar koja enkapsulira odgovarajuću Win32 kontrolu. Ova komponenta sadrži paletu alata, sa njenim kontrolama, i ima nešto veće mogućnosti.

Do sada ste već videli primere sa komponentom Toolbar u Poglavlju 5 kada smo razmatrali akcije. Da biste upotrebili ovu komponentu, potrebno je da je smestite na formular, a zatim upotrebite editor komponenata (iskačući meni se aktivira pritiskom na desni taster miša) da biste kreirali nekoliko kontrola i separatore. Primer komponente Toolbar za vreme izrade možete videti na slici 7.1.

Komponenta Toolbar se ispunjava objektima klase `TToolButton`. Ovo su interni objekti, baš kao što je `TMenuItem` interni objekat komponente MainMenu. Ovim objektima je osnovno svojstvo `Style`, koje određuje njihovo ponašanje:

- Stil `tbsButton` označava standardnu kontrolu koju možete da kliknete (push button).
- Stil `tbsCheck` označava kontrolu, čije ponašanje odgovara ponašanju polja za potvrdu, ili opcione kontrole, ukoliko je kontrola grupisana sa drugim kontrolama u bloku (kontrole su odvojene separatorima).



SLIKA 7.1 *Da biste kreirali paletu alata, možete odgovarajuću komponentu postaviti na formular, a zatim upotrebiti njen iskačući meni da biste dodali kontrole i separatore*

- Stil `tbsDropDown` označava kontrolu koja sadrži meni (drop-down button) koji je neka vrsta combo polja. Lista se u Delphiju može lako implementirati povezivanjem kontrole PopupMenu sa svojstvom `DropdownMenu` kontrole.
- Stilovi `tbsSeparator` i `tbsDivider` označavaju separatore sa ili bez različitih vertikalnih linija (u zavisnosti od svojstva `Flat` palete alata).

Da biste kreirali grafičku paletu alata, formularu možete dodati komponentu ImageList, učitati neke bitmape u komponentu, a zatim povezati ImageList sa svojstvom `Images` palete alata. Unapred je određeno da se slike dodeljuju kontrolama u redosledu u kojem se pojavljuju, ali ovakvo ponašanje možete veoma lako promeniti određivanjem svojstva `ImageIndex` svake od kontrola palete alata. Možete pripremiti buduće liste slika za specijalne uslove i dodeliti ih svojstvima `DisabledImages` i `HotImages` palete alata. Prva grupa se koristi za neaktivne kontrole, a druga se koristi za kontrolu koja se trentuno nalazi ispod pokazivača miša. Ovo je efekat koji je uveo Microsoft Internet Explorer.

Kod netrivijalnih aplikacija obično biste upotrebili komponentu ActionList, naročito ukoliko planirate da imate meni sa opcijama koje dupliraju kontrole palete alata (na primer, opcija sa menija File➥Save i kontrola Save). U tom slučaju biste kontrolama palete alata dodelili vrlo malo operacija, jer bi njihova svojstva i događaje obrađivale akcione komponente. Na primer, možete imati kontrolu palete alata koja menja ponašanje između "selektovano" i "nije selektovano" kao da je u pitanju polje za potvrdu. Ovakvo ponašanje dobijate menjanjem vrednosti `Checked` svojstva akcije svaki put kada se izvrši akcija. U tom slučaju nije potrebno podesiti kontrolu palete alata stilom `tbsChecked` jer će kod odrediti zahtevano ponašanje.

Toolbar i ActionList editora

U primeru MdEdit1 ja sam izradio meni i paletu alata koristeći kontrolu RichEdit, dajući prvi korak u izradi RTF (Rich Text File) editora koji ću dalje proširiti u ovom i narednim poglavljima.

Aplikacija je zasnovana na komponenti ActionList, koja sadrži akcije kojima se rukuje fajlovima i podršku za Clipboard, a može rukovati atributima fonta i paragrafa. Moj cilj nije izrada potpunog upotrebnog editora niti prikazivanje svake karakteristike kontrole RichEdit. Ja jednostavno želim da Vam pokažem kako da izradite korisnički interfejs programa, a za tu svrhu je korisno raditi sa upotrebnim primerom. Umesto da diskutujem o svim karakteristikama programa, ja ću samo istaći teme koje se odnose na trenutno razmatranje. Za detaljniji opis koda možete otvoriti dokument MdEdit Basics RTF koji se nalazi uz izvorni kod projekta.

Paleta alata primera MdEdit1 ima većinu svojih kontrola povezanih sa akcijama, koje su dostupne u samo jednoj ActionList komponenti koja se koristi za obradu svih elemenata menija. Samo poslednja kontrola, koja ima stil `tbsDropDown`, se obrađuje direktno a ne preko akcije. Evo strukture palete alata:

```
object ToolBar1: TToolBar
  AutoSize = True
  Flat = True
  Images = Images
  object ToolButton1: TToolButton
    Action = acNew
  end
```

```
  object ToolButton2: TToolButton
    Action = acOpen
  end
  ...
  object ToolButton11: TToolButton
    DropdownMenu = SizeMenu
    ImageIndex = 13
    Style = tbsDropDown
    OnClick = ToolButton11Click
  end
end
```

Poslednja kontrola je povezana sa PopupMenu komponentom (nazvanom `SizeMenu`). To je sve što je potrebno učiniti da biste prikazali spisak elemenata kada se selektuje strelica nadole, kao što možete videti na slici 7.2. Pošto se na kontrolu može i kliknuti, ja sam obezbedio obradu događaja, čime se povećava veličina selektovanog teksta.

SLIKA 7.2 *Paleta alata primera MdEdit1 sadrži kontrolu sa menijem (drop-down button) povezanu sa iskačućim menijem*

Tri kontrole za poravnanje paragrafa imaju svojstvo `Grouped` podešeno na vrednost `True`, čime se formira grupa (kao što se nalaze između dva separatora). Ovo je neophodno jer program proverava akciju koja odgovara trenutnom stilu, u događaju `OnUpdate` liste akcija, ali ne može da onemogući ostale dve akcije. Ponašanje korisničkog interfejsa elemenata menija se određuje stilom `RadioItem` kao i stilom kontrola palete alata koje su grupisane i svojstvom `AllowAllUp`.

IZRADA PALETA ALATA UPOTREBOM PANELA

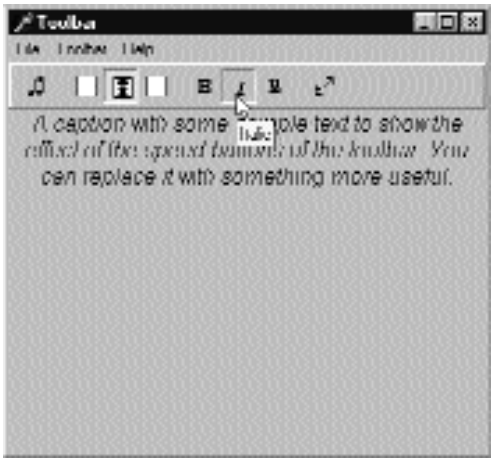
Pre nego što je kontrola Toolbar postala dostupna u Delphiju, standardni pristup pri izradi palete alata je bila upotreba panela koji je poravnat sa gornjom ivicom formulara i na koji je postavljen veliki broj komponenata SpeedButton. Speed button je malo zahtevan grafički element (prema Windows izvorima); ova komponenta ne može da dobije ulazni fokus, nema tab redosled i može se brže kreirati i iscrtati od bitmape.

Speed button se može ponašati kao push kontrola, polje za potvrdu ili opcione kontrole, a one mogu imati različite bitmape u zavisnosti od njihovog statusa. Da bi grupa speed button funkcionisala kao opcione kontrole, potrebno je da samo smestite nekoliko speed button kontrola na panel, sve ih

selektujete i dodelite istu vrednost svojstvu `GroupIndex` svake od kontrola. Sve kontrole koje imaju istu vrednost svojstva `GroupIndex` se međusobno isključuju. Jedna od ovih kontrola bi trebalo da uvek bude selektovana, te ne zaboravite da za svojstvo `Down` odredite vrednost `True` jedne od kontrola u vreme dizajniranja ili odmah pri pokretanju programa.

Određivanjem vrednosti za svojstvo `AllowAllUp` možete kreirati grupu međusobno isključivih kontrola, od kojih svaka može biti neaktivna — to jest, možete kreirati grupu iz koje korisnik može da odabere jednu opciju, ili se može desiti da ne odabere ni jednu. U specijalnom slučaju možete učiniti da speed button funkcioniše kao polje za potvrdu jednostavnim definisanjem grupe (svojstvo `GroupIndex`) koja sadrži samo jednu kontrolu i koja omogućava da kontrola ne bude selektovana (svojstvo `AllowAllUp`).

Konačno, možete odrediti vrednost `True` za svojstvo `Flat` svih komponenata SpeedButton, čime ćete dobiti savremeniji korisnički interfejs. Ukoliko ste zainteresovani za ovakav pristup, možete pogledati primer PanelBar koji je ovde ilustrovan.



Upotreba kontrola SpeedButton postaje manje uobičajena. Pored činjenice da je kontrola Toolbar veoma pogodna i definitivno predstavlja standard, kontrola SpeedButton ima dva velika problema. Prvi je da svaka od kontrola zahteva specifičnu bitmapu i ne može koristiti bitmapu iz liste slika (izuzev ukoliko ne napišete složen kod). Drugi je da kontrole SpeedButton ne funkcionišu najbolje uz akcije, jer neka svojstva, kao što je svojstvo Down, ne mapiraju direktno.

Combo polje na paleti alata

Ovaj primer možemo proširiti dodavanjem combo polja paleti alata. Brojne aplikacije koriste combo polja na paleti alata da bi se prikazale liste stilova, fontovi, veličine fontova i tako dalje. Pošto smo već koristili kontrolu sa menijem za veličinu fonta, možemo dodati combo polje kojim ćemo omogućiti brz izbor fonta. Ovo je jednostavno postići jer je kontrola Toolbar kontejnerska kontrola sa svim karakteristikama; možete direktno uzeti polje za izmene, combo

polje ili neku drugu kontrolu i smestiti je na paletu alata. Slika 7.3 prikazuje aplikaciju MdEdit2 koja sadrži combo polje za izbor fonta.

SLIKA 7.3 *Primer MdEdit2 u vreme izvršavanja*

Combo polje palete alata se inicijalizuje metodom `FormCreate`, koji izdvaja ekranske fontove koji su dostupni na sistemu:

```
ComboFont.Items := Screen.Fonts;
ComboFont := ComboFont.Items.IndexOf (
  RichEdit.Font.Name);
```

Combo polje inicijalno prikazuje naziv unapred određenog fonta koji se koristi u kontroli RichEdit, a koji se određuje u vreme dizajniranja. Ova vrednost se preračunava svaki put kada se trenutna selekcija promeni, koristeći font selektovanog teksta:

```
procedure TFormRichNote.RichEditSelectionChange (Sender: TObject);
begin
  ComboFont.ItemIndex :=
    ComboFont.Items.IndexOf (RichEdit.SelAttributes.Name);
end;
```

Kada se u combo polju odabere novi font, obavlja se obratna akcija. Tekst trenutnog elementa combo polja se dodeljuje kao naziv fonta za bilo koji selektovani tekst kontrole RichEdit:

```
procedure TFormRichNote.ComboFontClick (Sender: TObject);
begin
  RichEdit.SelAttributes.Name :=
    ComboFont.Text;
end;
```

Oblačići palete alata

Još jedan uobičajeni element paleta alata je *fly-by hint*, koji se još naziva i oblačić (balloon help) — tekst koji kratko opisuje kontrolu koja se trenutno nalazi ispod pokazivača miša. Ovaj tekst se obično prikazuje u žutom polju pošto se pokazivač miša zadrži neko vreme iznad kontrole. Da biste dodali oblačiće paleti alata aplikacije, jednostavno dodelite vrednost `True` svojstvu `ShowHints`.

Ja želim da `Caption` svake akcije upotrebim za oblačić te jednostavno mogu da ih kopiram u vreme izvršavanja umesto da ih odredim u vreme dizajniranja. Problem je što Caption koristi

karakter ampersand (&) koji se koristi za tastaturne prečice. Ovaj problem možemo rešiti uklanjanjem tog karaktera funkcijom `StripHotKey` u jedinici Menus. Evo koda:

```
procedure TFormRichNote.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  ...
  // move captions to hints, removing the &
  for I := 0 to ActionList.ActionCount - 1 do
    (ActionList.Actions[I] as TActionyHint :=
      StripHotKey ((ActionList.Actions[I] as TAction).caption);
end;
```

Kao što možete videti na slici 7.4, oblačići mogu sadržati string koji pokazuje tastaturne prečice koje su dodeljene svakom elementu menija kao podsetnik korisniku. Ovo je unapred određeno ponašanje koje možete onemogućiti određivanjem vrednosti svojstva `HintShortCuts` objekta `Application`. Ovaj globalni objekat kontroliše oblačiće i ostalim svojstvima i nekim metodima i događajima. Na primer, možete da promenite svojstva `HintColor`, `HintPause`, `HintHidePause` i `HintShortPause`. Primer MdEdit2 omogućava korisniku da prilagodi pozadinu oblačića izborom određenog elementa menija (Options➥Hint Color) sledećom obradom događaja:

```
procedure TFormRichNote.acHintColorExecute (Sender: T0bject);
begin
  ColorDialog.Color := Application.HintColor;
  if' ColorDialog.Execute then
    Application.HintColor := ColorDialog.Color;
end;
```

SLIKA 7.4 *Oblačići koje prikazuje primer MdEdit2*

NAPOMENA

Alternativno možete da promenite boju teksta oblačića obradom svojstva `OnShowHint` objekta `Application`. Ova obrada može da promeni boju teksta samo za određenu kontrolu. Događaj OnShowHint se koristi u narednom primeru CustHint. ■

Prilagođavanje oblačića

Kao što smo dodali oblačiće paleti alata formulara, sada možemo da dodamo oblačiće formularu ili komponentama formulara. Kada su u pitanju velike kontrole, oblačić će se pojaviti pored pokazivača miša. U nekim slučajevima je važno znati da će program prilagoditi kako se oblačići prikazuju.

Najjednostavnija stvar koju možete učiniti je promena vrednosti svojstva `HintColor` objekta `Application` (kao u prethodnom primeru) i tri svojstva koja se odnose na pauzu oblačića: `HintPause`, `HintHidePause` i `HintShortPause`. Prvo svojstvo definiše koliko dugo pokazivač miša treba da ostane iznad komponente pre nego što se prikaže oblačić, drugo svojstvo određuje koliko dugo će oblačić biti prikazan, a treće koliko dugo sistem treba da čeka da bi prikazao oblačić ukoliko je neposredno pre toga bio prikazan neki drugi oblačić.

Da biste imali više kontrole nad oblačićima, možete ih još više prilagoditi dodeljivanjem metoda događaju aplikacije `OnShowHint`. Potrebno je da ih ručno dodate ili — još bolje — da formularu dodate komponentu ApplicationEvents i obradite događaj komponente `OnShowHint`.

Metod koji ste definisali sadrži neke interesantne parametre, kao što su tekst stringa oblačića, `Boolean` zastavica za aktiviranje oblačića i struktura daljih informacija:

```
TShowHintEvent = procedure (
  var HintStr: string;
  var CanShow: Boolean;
  var HintInfo: THintInfo) of object;
```

Svaki od parametara se prosleđuje po referenci tako da imate mogućnost da ih promenite. Poslednji parametar je struktura koja sadrži referencu na kontrolu, poziciju oblačića i druge informacije:

```
THintInfo = record
  HintControl: TControl;
  HintPos: TPos;
  HintMaxWidth: Integer;
  HintColor: TColor;
  CursorRect: TRect;
  CursorPos: TPoint;
end;
```

Vi možete da izmenite informacije ove strukture; na primer, možete da promenite poziciju oblačića pre prikazivanja. To sam ja učinio u primeru CustHint koji prikazuje oblačić za oznaku u centralnoj oblasti. Evo šta možete da napišete da biste prikazali oblačić za veliku oznaku na sredini njene površine:

```
procedure TForm1.ShowHint (var HintStr: string;
  var CanShow: Boolean; var HintInfo: THintInfo);
begin
  with HintInfo do
    if HintColor = Label1 then
      HintPos := HintControl.ClientToScreen (Point (
        HintControl.Width div 2, HintControl.Height div 2));
end;
```

Kod treba da proračuna centar generičke kontrole (`HintInfo.HintControl`) i da zatim konvertuje koordinate kontrole u ekranske koordinate, primenjujući metod `ClientToScreen` same kontrole.

Možemo da unapredimo primer CustHint i na drugi način. Kontrola RadioGruop formulara sadrži tri opcione kontrole. Ipak, to nisu samostalne komponente već jednostavno klonovi opcione kontrole koji su iscrtani na površini kontrole RadioGroup. Šta se dešava ukoliko želimo da dodamo oblačić za svaku od opcionih kontrola?

Polje `CursorRect` sloga `THintInfo` može da posluži u tu svrhu. Ono označava oblast komponente preko koje se može pomerati pokazivač miša, a da se ne deaktivira oblačić. Kada se pokazivač miša pomeri van oblasti, Delphi sakriva prozor oblačića. Ukoliko navedemo različit tekst oblačića i različitu oblast za svaku od opcionih kontrola, možemo praktično obezbediti tri različita oblačića. Budući da proračunavanje stvarne pozicije svake od opcionih kontrola nije lako, ja sam podelio površinu kontrole RadioGroup u mnogo jednakih delova kao da postoje opcione kontrole. Tekst opcione kontrole (ne selektovani element već element koji se nalazi ispod pokazivača miša) se zatim dodaje tekstu oblačića:

```
procedure TForm1.ShowHint (var HintStr: string;
  var CanShow: Boolean; var HintInfo: THintInfo);
var
  RadioItem, RadioHeight: Integer;
  RadioRect: TRect;
begin
  with HintInfo do
  if HintControl = Labe11 // as before
  else
  if HintControl = RadioGroupl then
  begin
    RadioHeight (RadioGroup1.Height) div
      RadioGroup1.Items.Count;
    RadioItem := CursorPos.Y div RadioHeight;
    HintStr := 'Choose the ' +
      RadioGroup1.Items [RadioItem] + 'button';
    RadioRect := RadioGroup1.ClientRect;
    RadioRect.Top := RadioRect.Top +
      RadioHeight * RadioItem;
    RadioRect.Bottom := RadioRect.Top + RadioHeight;
    // assign the hints rect and pos
    CursorRect := RadioRect;
  end;
end;
```

Poslednji deo koda priprema pravougaonik oblačića, počevši od pravougaonika koji odgovara klijent površini komponente i promenom vrednosti `Top` i `Left` na odgovarajuću sekciju komponente `RadioGroup`. Rezultujući efekat je da svaka opciona kontrola za RadioGroup izgleda kao da ima sopstveni oblačić, kao što se može videti na slici 7.5.

SLIKA 7.5 *Kontrola RadioGroup primera CustHint prikazuje različite oblačiće već prema opcionoj kontroli iznad koje se nalazi pokazivač miša*

Kontejneri palete sa alatima

Većina savremenih aplikacija sadrži više paleta alata koje obično podržava određeni kontejner. Microsoft Internet Explorer, različite standardne poslovne aplikacije i Delphi IDE koriste ovaj opšti pristup. Ipak, svaka od ovih aplikacija ovaj koncept različto primenjuje. Delphi sadrži dva kontejnera palete alata koja su odmah spremna za upotrebu; to su komponente CoolBar i ControlBar. U njihovom korisničkom interfejsu postoje razlike, ali je najveća u tome što je CoolBar Win32 kontrola, dakle, deo operativnog sistema, dok je ControlBar VCL komponenta.

Obe komponente mogu da sadrže kontrole palete alata kao i neke druge elemente, kao što su combo polja i druge kontrole. Zapravo, paleta alata može da zameni meni aplikacije, što ćemo kasnije videti.

Ove dve komponente ćemo razmatrati u naredna dva odeljka, ali ovde želim da naglasim (a da ne idem suviše unapred) da ja obično koristim ControlBar. Ova komponenta je zasnovana na VCL-u (i nije podložna promenama sa svakom manjom verzijom Microsoft Internet Explorera), a njen korisnički interfejs je lepši i više podseća na uobičajene aplikacije.

Zaista lepa paleta alata

Komponenta CoolBar je u osnovi kolekcija objekata `TCoolBand`. Za razliku od kontrola palete alata, ovi objekti se ne javljaju kao samostalni objekti formulara već jednostavno kao kolekcije podelemenata. U Object Inspectoru se pojavljuju samo kada selektujete editor CoolBar svojstva `Bands`, kao što možete da vidite na slici 7.6. Možete da kreirate jednu ili više traka i da zatim odredite njihove atribute.

SLIKA 7.6 *Editor svojstava svojstva Bands komponente CoolBar funkcioniše zajedno sa Object Inspectorom*

Komponentu CoolBar možete da prilagodite na mnogo načina. Možete da odredite bitmapu koja će poslužiti kao njena pozadina, možete da dodate trake koristeći editor svojstva `Bands`, a zatim svakoj traci dodelite postojeću komponentu ili kontejner komponente. Možete da upotrebite bilo koju kontrolu koja ima svoj prozor (ne grafičke kontrole), ali će se samo neke pravilno prikazati. Ukoliko želite da imate bitmapu u pozadini za CoolBar, na primer, potrebno je da koristite delimično transparentne kontrole.

Tipična komponenta koja se koristi u CoolBaru je komponenta Toolbar (koja se može učiniti potpuno transparentnom), ali su i combo polja, polja za izmene i kontrole za animaciju takođe veoma česti. Ovo je često inspirisano korisničkim interfejsom Internet Explorera, prve Microsoftove aplikacije koja je upotrebila komponentu CoolBar.

Možete da smestite jednu traku u jednu liniju ili ih sve smestite u jednu liniju. Svaka od njih bi koristila deo moguće površine i automatski bi bila uvećana kada korisnik klikne njen naslov. Lakše je koristiti novu komponentu nego je objasniti. Pokušajte sami da je isprobate ili pratite opis koji ću dati, a kojim ćemo izraditi novu verziju našeg primera sa paletom alata, koji se zasniva na kontroli CoolBar. Formular koji prikazuje aplikacija u vreme izvršavanja možete videti na slici 7.7.

SLIKA 7.7 *Formular primera CoolBar u vreme izvršavanja*

Primer CoolBar sadrži komponentu `TCoolBar` sa četiri trake, po dve u svakoj od dve linije. Prva linija sadrži podskup palete alata prethodnog primera, ali ovoga puta sa dodatkom ImageList za

označene slike. Druga sadrži polje za izmene koje se koristi za podešavanje fonta teksta. Treća sadrži komponentu ColorGrid koja se koristi za izbor boje fonta i boje pozadine fonta. Poslednja traka sadrži kontrolu ComboBox u kojoj su prikazani fontovi koje možete da upotrebite.

ControlBar

Korisnički interfejs komponente CoolBar je zaista veoma privlačan i Microsoft ga sve više koristi u svojim aplikacijama. Ipak, Windows kontrola CoolBar postoji u mnogo različitih i međusobno nekompatibilnih verzija jer je Microsoft napisao različite verzije biblioteke kontrola za različite verzije Internet Explorera. Neke od ovih verzija su postojeće programe izrađene u Delphiju učinile neupotrebljivim.

NAPOMENA

Interesantno je da Microsoftove aplikacije, uopšte uzev, ne koriste biblioteke kontrola. Word i Excel koriste svoje interne verzije kontrola dok VB koristi OCX, a ne direktno kontrole. Deo razloga zbog kojih Borland ima mnogo više problema sa kontrolama je taj što ih on više koristi (i na više načina) nego što to čak čini Microsoft. ■

Zbog toga je Borland (u Delphiju 4) predstavio kontejner palete alata nazvan ControlBar. Linija kontrola može da sadrži nekoliko kontrola, kao što je to slučaj sa CoolBarom, i nudi sličan korisnički interfejs koji korisnicima omogućava da prevlače elemente i prepoznaju paletu alata u vreme izvršavanja. Dobar primer upotrebe kontrole ControlBar je Delphijeva sopstvena paleta alata, međutim, Microsoftove aplikacije koriste veoma sličan korisnički interfejs.

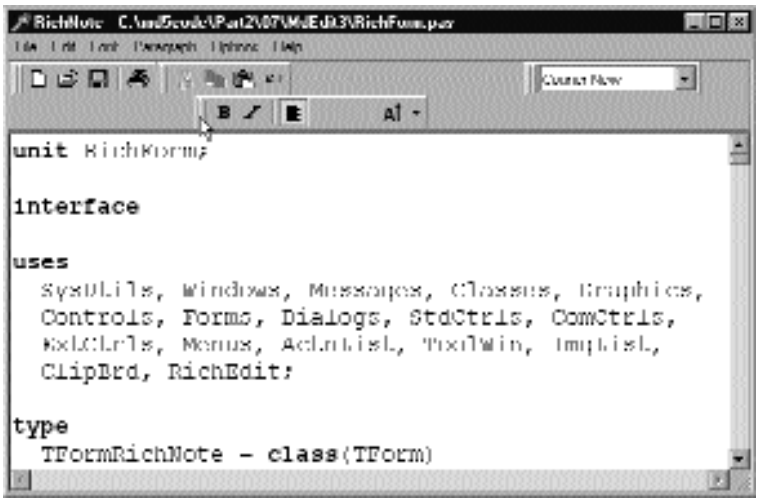
ControlBar je kontejner kontrola i formirate ga tako što na njega postavljate druge kontrole kao što biste to učinili sa panelom. Svaka kontrola koja je smeštena na liniju dobija svoje područje prevlačenja (mali panel sa dve vertikalne linije na levoj strani kontrole), kao što možete videti na slici 7.8. Zbog toga bi trebalo da izbegavate smeštanje specifičnih kontrola unutar ControlBara i da više dodajete kontejnere sa kontrolama unutar njih. Umesto upotrebe panela trebalo bi da koristite jednu kontrolu Toolbar za svaki odeljak palete alata.

SLIKA 7.8 *ControlBar je kontejner koji korisniku omogućava da prevlači sve elemente upotrebom specijalne linije za prevlačenje. Primetite da svaka kontrola dobija zasebnu liniju prevlačenja, ono što obično želite da izbegnete.*

Primer MdEdit3 je još jedna verzija demonstracionog programa RichEdit koji razvijamo u ovom poglavlju. Ja sam u osnovi grupisao kontrole u tri palete alata (umesto u samo jednu), a combo polje sam ostavio posebno. Sve ove komponente su unutar ControlBara tako da korisnik može da ih uredi po želji, što se može videti na slici 7.9 i u sledećem DFM listingu:

```
object CantrolBar1: TControlBar
  Align = alTop
  AutoSize = True
  ShowHint = Tru   e
  object ToolBarFile: TToolBar
    AutoSize = True
    EdgeBorders = []
    EdgeInner = esNone
    EdgeCuter = esNone
    Flat = True
    Images = Images
    Wrapable = False
    object ToolButton1: TToolButton
      Action = acNew
    end
    // more buttons
  end
  object ToolBarEdit: TToolBar
    // similar properties
    object ToolButton6: TToolButton
      Action = acCut
    end
    // more buttons
  end
  object ToolBarFont: TToolBar
    // ...
  end
  object ComboFont: TComboBox
    Hint = 'Font Family'
    Style = csDropDownList
    Font.Height = -11
    FontName = 'Anal'
    ItemHeight = 14
    ParentFont = False
    Sorted = True
    OnClick = ComboFontClick
  end
end
```

Primetite u listingu da morate da isključite ivice za kontrole palete alata da biste dobili standardni efekat i da biste dobili ravan stil. Podešavanje veličine svih kontrola, tako da postignete iste visine jednog ili dva reda elemenata, nije lako kao što može da izgleda na prvi pogled. Za neke kontrole veličina se određuje automatski ili imaju različita ograničenja. Da biste učinili da combo polje bude iste visine kao i paleta alata, morate da "ugurate" tip i veličinu fonta unutar polja. Promena veličine same kontrole nema efekta.



SLIKA 7.9 *Primer MdEdit3 u vreme izvršavanja kada korisnik menja uređenje paleta alata unutar ControlBara*

ControlBar takođe sadrži meni sa prečicama koje Vam omogućavaju da prikažete ili sakrijete svaku od kontrola koja se nalazi unutar ControlBara. Umesto da napišem specifičan kod za ovaj primer, ja sam upotrebio više generičko rešenje (koje se može ponovo upotrebiti). Meni sa prečicama, nazvan `BarMenu`, je prazan u vreme dizajniranja, a popunjava se kada se program pokrene:

```
procedure TFormRichNote.FormCreate(Sender: TObject);
var
  I: Integer;
  mItem: TMenuItem;
begin
  ...
  // populate the control bar menu
  for I := 0 to ControlBar.ControlCount - 1 do
  begin
    mItem := TMenuItem.Create (Self);
    mItem.Caption := ControlBar.Controls [I].Name;
    mItem.Tag := Integer (ControlBar.Controls [I]);
    mItem.OnClick := BarMenuClick;
    BarMenu.Items.Add (mItem);
end;
```

Procedura `BarMenuClick` je obrada događaja koju koriste svi elementi menija i koja koristi `Tag` svojstva elementa menija `Sender` da bi se referisala na element ControlBara koji je povezan sa elementom `FormCreate` metoda:

```
procedure TFormRichNote.BarMenuClick (Sender: TObject);
var
  aCt rl: TControl;
begin
  aCtrl := TControl ((Sender as TComponent).Tag);
  aCtr1.Visible := not aCtrl.Visible;
end;
```

Konačno, događaj OnPopup menija se koristi za obnavljanje oznaka elemenata menija:

```
procedure TFormRichNote.BarMenuPopup(Sender: TObject);
var
  I: Integer;
begin
  // update the menu checkmarks
  for I := 0 to BarMenu.Items.Count - 1 do
    BarMenu.Items [I].Checked :=
      TControl (BarMenu.Items [I] .Tag).Visible;
end;
```

Meni u ControlBaru

Ukoliko pogledate korisnički interfejs Delphi okruženja za programiranje, možete videti da ControlBar takođe sadrži i meni aplikacije, koji se može prevlačiti na isti način kao i palete alata i Component Palette. Kako možemo da dodamo meni ControlBaru naše aplikacije?

Meni formulara se ne može smestiti unutar ControlBara, ali mi možemo da dodamo još jednu kontrolu palete alata koja će sadržati meni. Ova kontrola treba da sadrži svojstvo `ShowCaptions` i treba da bude određena vrednost `True` za svojstvo `Flat`. Zatim bi trebalo da dodate onoliko kontrola alata koliko postoji menija, da za njihova svojstva `AutoSize` i `Grouped` odredite vrednost `True` i da povežete svaku kontrolu alata sa odgovarajućim menijem koristeći svojstvo `MenuItem`.

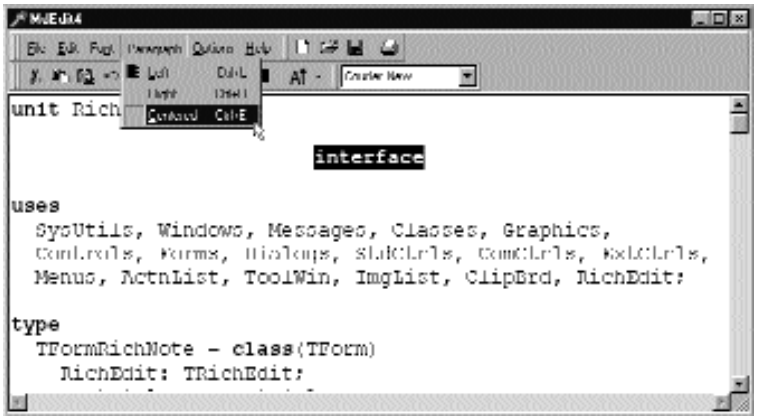
SAVET

Borland je načinio besplatnu komponentu TMenuBar koju možete da preuzmete sa njihovog web sajta (iz Delphi sekcije Downloads). Ova komponenta se direktno povezuje sa komponentom MainMenu i automatski određuje potrebna svojstva. ■

Ponoviću još jednom; umesto da sve ove operacije obavite u vreme dizajniranja, možete da automatizujete kreiranje onoliko kontrola koliko je potrebno za meni, dodajući kod metodu `FormCreate`:

```
// create the buttons of the menu toolbar
ToolSize = 0;
for I := MainMenu.Items.Count - 1 downto 0 do
begin
  tb := TToolButton.create (ToolBarMenu)j;
  tb.Parent := ToolBarMenu;
  tb.Autosize := True;
  tb.Grouped := True;
  tb.Caption := MainMenu.Items[I].Caption;
  tb.MenuItem := MainMenu.Items[I];
  Inc (ToolSize, tb.Width);
end;
// size the menu toolbar
ToolBarMenu.Wwidth := ToolSize;
// hide the standard menu, using the form's Menu property
Menu := nil;
```

Primetite da je potrebno da isključite meni sa formulara uklanjanjem vrednosti svojstva formulara `Menu`, koje je automatski određeno prilikom postavljanja komponente menija na formular. Rezultat je meni unutar ControlBara, kao što možete videti na slici 7.10.



SLIKA 7.10 *Primer MdEdit4 prikazuje kako da smestite meni unutar palete alata koja se zasniva na komponenti ControlBar*

Kreiranje statusne linije

Izrada statusne linije je još jednostavnija od izrade palete alata. Delphi sadrži komponentu StatusBar, koja se zasniva na odgovarajućoj Windows kontroli. Ova kontrola može da se koristi gotovo kao panel kada je za svojstvo `SimplePanel` određena vrednost `True`. U tom slučaju možete da upotrebite svojstvo `SimpleText` da biste prikazali tekst. Stvarna prednost ove komponente je da omogućava da definišete veliki broj potpanela aktiviranjem njenog svojstva `Panels`. (Ovaj editor svojstava možete prikazati tako što ćete dva puta kliknuti kontrolu StatusBar.) Svaki od potpanela ima svoje grafičke atribute koji se mogu prilagoditi upotrebom editora. Još jedna karakteristika komponente StatusBar je oblast "size grip" koja je dodata u donji levi ugao linije, a koja je korisna za promenu veličine samog formulara. Ovo je tipičan element Windows korisničkog interfejsa, a možete ga kontrolisati svojstvom `SizeGrip`.

Postoje mnogi načini da se upotrebi statusna linija. Najčešći način je prikazivanje informacija o elementu menija koji je trenutno selektovao korisnik. Pored ovoga, statusna linija često prikazuje i druge informacije o statusu programa: poziciju kursora kod grafičkih aplikacija, trenutnu liniju teksta kod tekst-procesora, status tastera, datum i vreme i tako dalje.

Oblačići menija na statusnoj liniji

Nova verzija editora, MdEdit5, sadrži statusnu liniju koja može da prikazuje opise elemenata menija, status tastera Caps Lock i trenutnu poziciju kursora. Komponenta StatusBar ovoga primera sadrži četiri panela. Mada ćemo tekst prikazivati na samo tri od njih, potrebno je da definišemo i četvrti da bismo odvojili oblast trećeg panela. Poslednji panel je uvek dovoljno veliki da prekrije preostalu površinu statusne linije.

Da biste prikazali informacije na panelu, jednostavno upotrebite njegovo svojstvo `Text` u izrazu kao što je ovaj:

```
StatusBar1.Panels[1] := 'message';
```

Paneli nisu nezavisne komponente tako da ne možete da im pristupite po nazivu. Dobro rešenje kojim se povećava čitljivost programa je definisanje konstante za svaki panel koji želite da koristite, a zatim koristite te konstante kada se referišete na panele. Primer MdEdit5 definiše sledeće konstante:

```
const
  sbpMessage = 0;
  sbpCaps = 1;
  sbpPositin = 2;
```

Sada je potrebno da popunimo panele statusne linije odgovarajućim tekstom. Prvo, želimo da prikažemo poruku za elemente menija i kontrole palete alata. Da bismo ostvarili ovaj efekat, potrebno je da izvršimo dva koraka. Najpre unesite string za svojstvo `Hint` za svaku akciju komponente ActionList. Ovaj string će se koristiti za oblačić kontrola palete alata i kao poruka na statusnoj liniji kada se pokazivač miša nađe iznad kontrole, ili kada se odabere element menija. Zapravo, mi možemo da upotrebimo svojstvo `Hint` da bismo naveli različite stringove za dva slučaja, tako što ćemo uneti string koji je podeljen na dva dela separatorom |. Na primer, možete uneti sledeći string kao vednost svojstva `Hint`:

```
'Help|Activate the help of the application'
```

Prvi deo stringa, *Help*, se koristi za oblačiće, dok se drugi deo stringa prikazuje na statusnoj liniji. Primer ovog efekta možete videti na slici 7.11.

SAVET

Kada se oblačić kontrole sastoji od dva stringa, možete da upotrebite metode `GetShortHint` i `GetLongHint` da biste izdvojili prvi (kraći) i drugi (duži) podstring iz stringa koji prosleđujete kao parametar, što je obično vrednost svojstva Hint. ■

SLIKA 7.11 *Statusna linija primera MdEdit5 prikazuje (između ostalih informacija) opis trenutne kontrole ili elementa menija. Dva dela svojstva Hint se prikazuju na statusnoj liniji i u oblačiću.*

Da biste prikazali tekst na statusnoj liniji, potrebno je da napišete kod kojim se obrađuje događaj aplikacije `OnHint`. Da bismo izbegli ručno dodavanje novog metoda formularu, a zatim njegovo dodeljivanje događaju `OnHint` objekta `Applicaton`, možemo formularu dodati komponentu ApplicationEvents i obraditi ovaj događaj u vreme izvršavanja.

Procedura `ShowHint` kopira na statusnu liniju trenutnu vrednost svojstva `Hint` aplikacije, koje privremeno sadrži kopiju stringa trenutno selektovanog elementa:

```
procedure TFormRichNote.Showhint (Sender: TObject);
begin
  StatusBar1.Panels[sbpMessage].Text := Application.Hint;
end;
```

Ovo je sve što je potrebno da bi se prikazao string koji opisuje efekat menija na statusnoj liniji.

Da biste prikazali status tastera Caps Lock, ili bilo kojeg drugog tastera, potrebno je da pozovete API funkciju `GetKeyState` koja kao rezultat daje broj. Ukoliko je postavljen niži bit ovog broja (to jest, ukoliko je broj neparan), tada je taster pritisnut. Kada proveravamo status? Mi to možemo da učinimo svaki put kada korisnik pritisne taster, kada aplikacija ništa ne obavlja, ili možemo dodati tajmer i vršiti proveru svakih 5 sekundi. Ovaj drugi pristup ima prednost jer korisnik može da pritisne taster Caps Lock prilikom rada sa drugom aplikacijom, a to treba da bude naznačeno i na statusnoj liniji našeg programa. Ipak, upotreba tajmera usporava reakciju na pritisak tastera, dok smanjivanje vremena tajmera može da uspori aplikaciju. Dakle, ja sam odlučio da napišem jednostavnu proceduru, koju sam nazvao `CheckCapsLock`, i da je pozovem u obradi događaja `OnUpdate` komponente ActionList (koja se poziva kada aplikacija niša ne obavlja) i u obradi događaja `OnTimer` komponente tajmer koju sam dodao formularu:

```
procedure TFormRichNote.CheckCapslock;
begin
  if odd (GetKeyState (VK_CAPITAL) then
    StatusBar1.Panels[sbpCaps].Text := 'CAPS'
  else
    StatusBar1.Panels[sbpCaps].Text := '';
end;
```

Konačno, program koristi treći panel za prikazivanje pozicije kursora (merene linijama i karakterima po liniji) svaki put kada se selekcija promeni. Pošto su vrednosti `CaretPos` bazirane na nuli (to jest, gornji levi ugao linije je 0, karakter 0), ja sam odlučio da dodam jedan svakoj vrednosti da bih ih učinio upotrebljivim povremenom korisniku:

```
procedure TFormRichNote.RichEditSelectionChange (Sender: TObject);
begin
  ...
  // update the position in the status bar
  StatusBar.Panels[sbpPosition].Text := Format ('%d%d',
    [RichEdit.CaretPos.Y + 1, RichEdit.CarePos.X + 1]);
end;
```

Skrolovanje formulara

Kada izrađujete jednostavnu aplikaciju, jedan formular može da sadrži sve komponente koje su Vam potrebne. Kako aplikacija raste, možda je potrebno da zbijete komponente, povećate veličinu formulara ili dodate nove formulare.

Ukoliko smanjite površinu koju zauzimaju komponente, možete dodati mogućnost promene veličine komponenata u vreme izvršavanja, a moguće je i da podelite formular na dve oblasti. Ukoliko odlučite da povećate veličinu formulara, možete da upotrebite klizače da biste omogućili korisniku da se kreće u okviru formulara čija je veličina veća od veličine ekrana.

NAPOMENA

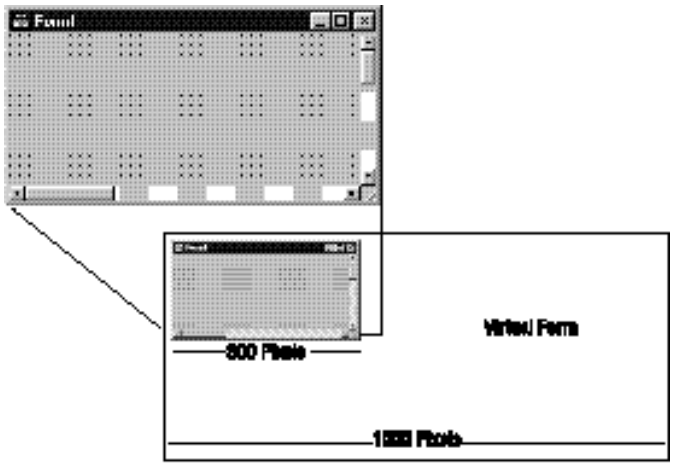
Ukoliko odlučite da dodate novi formular, možete kreirati sekundarne formulare i okvire za dijalog, kreirati formulare sa više strana ili upotrebiti MDI pristup (što ćemo opisati u narednom poglavlju). ■

Dodavanje klizača formularu je veoma jednostavno. Zapravo, nije potrebno ništa učiniti. Ukoliko smestite veliki broj komponenata na veliki formular i zatim mu smanjite veličinu, automatski će se dodati klizač, izuzev ukoliko niste promenili unapred određenu vrednost `True` svojstva `AutoScroll`.

Pored svojstva `AutoScroll`, formulari sadrže još dva svojstva, svojstva `HorzScrollBar` i `VertScrollBar`, koja mogu da se koriste za određivanje nekolikih svojstava objekta `TFormScrollBar` koja su povezana sa formularom. Svojstvo `Visible` označava da li je klizač prisutan, svojstvo `Position` određuje početni status kvadratića klizača, a svojstvo `Increment` određuje efekat kada se klikne jedna od strelica na krajevima klizača. Najvažnije svojstvo je, ipak, `Range`.

Svojstvo `Range` klizača određuje virtuelnu veličinu formulara u jednom smeru, a ne stvarni opseg vrednosti klizača. U početku ovo može da bude zbunjujuće. Evo primera koji bi trebalo da razjasni kako funkcioniše svojstvo `Range`. Pretpostavimo da Vam je potreban formular koji treba da sadrži veliki broj komponenata, te će zbog toga biti širok 1000 piksela. Mi možemo da upotrebimo ovu vrednost da odredimo "virtuelni opseg" formulara promenom opsega horizontalnog klizača. Pogledajte sliku 7.12 na kojoj je ilustracija virtuelne veličine formulara koja je nagoveštena opsegom klizača. Ukoliko je klijent oblast formulara manja od 1000 piksela, prikazaće se klizač. Sada možete da ga koristite u vreme dizajniranja da biste dodali nove komponente "sakrivenom" delu formulara.

Svojstvo `Position` klizača je iz opsega od 0 do 1000 minus trenutna veličina klijent oblasti. Na primer, ukoliko je klijent oblast širine 300 piksela, tada možete preći 700 piksela da biste videli šta se nalazi na kraju formulara (hijadu piksela).



SLIKA 7.12 *Reprezentacija virtuelne veličine formulara nagoveštena opsegom klizača*

Primer testiranja skrolovanja

Ja sam izradio primer Scroll koji sadrži virtuelni formular veličine 1000 piksela. Da bih ovo postigao, jednostavno sam podesio opseg horizontalnog klizača na 1000:

```
object Form1: TForm1
  Width = 458
  Height = 368
  HorzScrollBar.Range = 1000
  VertScrollBar.range = 305
  AutoScroll = False
  Caption = 'Scrolling Form'
  OnResize = FormResize
  ...
```

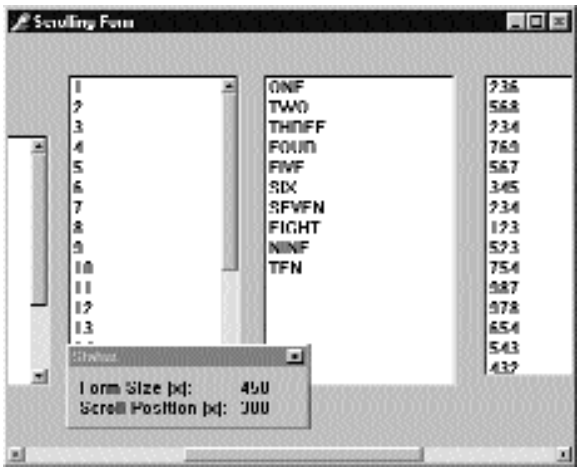
Formular ovog primera je popunjen velikim brojem besmislenih lista, a ja sam mogao da dobijem isti efekat opsega klizača da sam postavio poslednju listu na njenu poziciju (`Left`) na koju bih dodao veličinu (`Width`) da bih dobio 1000.

Interesantni deo primera je postojanje prozora palete alata koji prikazuje status formulara i njegovog horizontalnog klizača. Ovaj drugi formular sadrži četiri oznake: dve sa konstantnim tekstom i dve za izlaz. Pored toga, sekundarni formular (nazvan `Status`) ima stil bordure `bsToolWindow` i nalazi se iznad svih prozora. Takođe bi trebalo da odredite vrednost `True` za svojstvo `Visible` da bi se prozor automatski prikazao prilikom pokretanja:

```
object Status: TStatus
  BorderIcons = [biSystemMenu]
  BorderStyle = bsToolWindow
  Caption = 'Status'
  FormStyle = fsStayOnTop
  Visible = True
  object Label1: TLabel . .
  ...
```

Nema mnogo koda u programu. Cilj programa je da ažurira vrednosti u paleti alata svaki put kada se promeni veličina formulara ili kada se skroluje (kao što možete videti na slici 7.13). Prvi deo je neverovatno jednostavan. Možete da obradite događaj `OnResize` formulara i jednostavno kopirate nekoliko vrednosti u dve oznake. Oznake su deo drugog formulara tako da je potrebno da upotrebite prefiks `Status`, što je naziv instance formulara:

```
procedure TForm1.FormResize (Sender: TObject);
begin
  Status.Label3.Caption := IntToStr(ClientWidth);
  Status.Label4.Caption := IntToStr(HorizScrollBar.Position);
end;
```



SLIKA 7.13 *Izlaz primera Scroll1*

Da smo želeli da promenimo izlaz svaki put kada korisnik skroluje sadržaj formulara, tada ne bismo mogli da koristimo Delphi obradu događaja jer ne postoji događaj `OnScroll` za formulare (mada postoji za samostalne ScrollBar komponente). Izostavljanje ovog događaja ima smisla jer Delphi formulari na veoma moćan način automatski upravljaju klizačima. Nasuprot tome, u Windowsu su klizači elementi veoma niskog nivoa i zahtevaju mnogo kodiranja. Obrada događaja skrolovanja ima smisla u specijalnim slučajevima kao kada želite da precizno vodite računa o operacijama skrolovanja koje obavlja korisnik.

NAPOMENA

Još jednom, ono što mi se zaista dopada u Delphiju je da obrada Windows poruka koju ne podržava okruženje zahteva samo još jednu liniju koda. Ja nikada do sada nisam video nešto tako dobro u nekom drugom vizuelnom okruženju. ■

Evo koda koji je potrebno da napišemo. Prvo dodajte deklaraciju metoda klasi i povežite je sa Windows porukom horizontalnog klizača (`wm_HScroll`):

```
public
  procedure FormScroll (var ScrollData: TWMScroll);
    message wm_HScroll;
```

Zatim unesite kod ove procedure koji je gotovo identičan kodu metoda `FormResize` koji smo ranije videli:

```
procedure TForm1.FormScroll (var ScrollData: TWMScroll);
begin
  inherited;
  Status.Label3.Caption := IntToStr(ClientWidth);
  Status.Label4.Caption := IntToStr(HorizScrollBar.Position);
end;
```

Veoma je važno dodati poziv za ključnu reč `inherited`, kojom se aktivira metod koji se odnosi na istu poruku u osnovnoj klasi formulara. Ključna reč `inhereted` u Windows obradama poruka poziva metod osnovne klase koji zaobilazimo, a koji je povezan sa odgovarajućom Windows porukom (čak i ako je naziv procedure drugačiji). Bez ovog poziva formular neće imati svoje unapred određeno ponašanje prilikom skrolovanja, to jest, skrolovanje uopšte neće biti moguće.

Automatsko skrolovanje

Svojstvo klizača Range može da izgleda čudno sve dok konstantno ne počnete da ga upotrebljavate. Kada malo razmislite o njemu, uvidećete prednosti pristupa "virtuelnog opsega". Prvo, klizač se automatski uklanja sa formulara kada je klijent oblast formulara dovoljno velika da prikaže virtuelnu veličinu; kada smanjite veličinu formulara, ponovo se dodaju klizači.

Ova karakteristika je naročito interesantna onda kada je za svojstvo `AutoScroll` određena vrednost `True`. U tom slučaju, krajnje pozicije kontrole koja je krajnje desno i dole se automatski kopiraju u svojstva Range dva klizača formulara. Automatsko skrolovanje dobro funkcioniše u Delphiju. U poslednjem primeru virtuelna veličina formulara bi bila određena desnom granicom poslednje liste. Ovo je definisano sledećim atributima:

```
object ListBox6: TListBox
  Left = 832
  Width = 145
end
```

Zbog toga bi horizontalna virtuelna veličina formulara bila `977` (što predstavlja zbir gornje dve vrednosti). Ovaj broj se automatski kopira u polje `Range` svojstva formulara `HorzScrollBar`, izuzev ukoliko ga ručno ne promenite da biste imali veći formular (kao što sam ja to učinio u primeru Scroll, određujući vrednost 1000 da bih ostavio malo prostora između poslednje liste i bordure formulara). Ovu vrednost možete da odredite u Object Inspectoru, ili možete načiniti sledeći test. Pokrenite program, podesite veličinu formulara po želji i pomerite kvadratić klizača u krajnju desnu poziciju. Kada povećate veličinu formulara i pomerite kvadratić, uvek ćete dobiti 1000, virtuelnu koordinatu poslednjeg desnog piksela formulara, kada je formular bilo koje veličine.

Skrolovanje i koordinate formulara

Upravo smo videli da formulari mogu automatski skrolovati svoj sadržaj. Ali, šta se dešava ukoliko crtate direktno na površini formulara? Javljaju se neki problemi, ali je rešenje nadohvat ruke. Pretpostavimo da želimo da nacrtamo nekoliko linija na virtuelnoj površini formulara, kao što je prikazano na slici 7.14.

Kako verovatno ne posedujete monitor koji može da prikaže 2000 piksla po bilo kojoj osi, možete da kreirate manji formular, dodate dva klizača i pravilno odredite njihovo svojstvo Range kao što sam ja to učinio u primeru Scroll2. Evo tekstualnog opisa formulara:

```
object Form1: TForm1
  HorzScrollBar.Range = 2000
  VertScrollBar.Range = 2000
  ClientHeight = 336
  ClienWidth = 472
  OnPaint = FormPaint
end
```

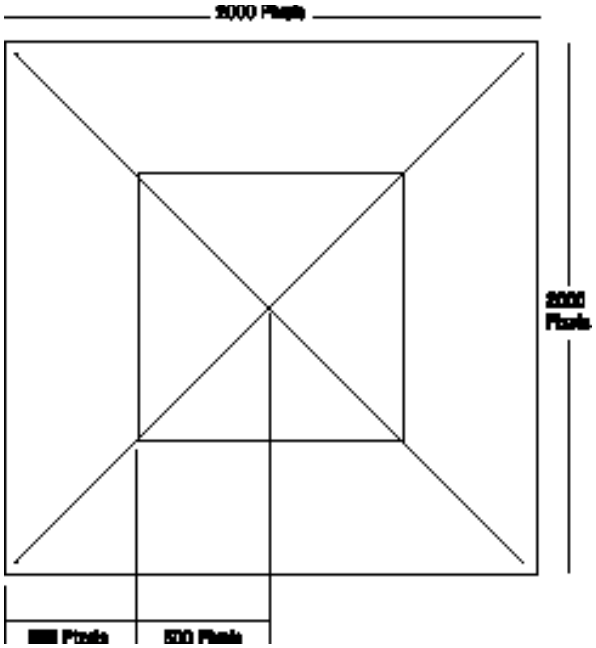
Ukoliko jednostavno nacrtamo linije koristeći virtuelne koordinate formulara, slika se neće pravilno prikazati. Zapravo, u odgovarajućem metodu `OnPaint`, potrebno je da sami proračunamo virtuelne koordinate. Na sreću, to je lako jer znamo da virtuelne koordinate `X1` i `Y1` gornjeg levog ugla klijent oblasti odgovaraju trenutnoj poziciji dva klizača:

```
procedure TForm1.FormPaint(Sender: TObject);
var
  X1, Y1: Integer;
begin
  X1 := HorzScrollBar.Position;
  Yl := VertScrollBar.Position;

  // draw a yellow line
  Canvas.Pen.Width := 30;
  Canvas.Pen.Color := clYellow;
  Canvas.MoveTo (30-X1, 30-Y1);
  Canvas.LineTo (l970-X1, l970-Y1);
// and so on . . .
```

Bolja alternativa, umesto proračunavanja koordinata za svaku operaciju izlaza, je da možemo pozvati `SetWindowOrgEx` API da bismo pomerili koordinatni početak samog `Canvasa`. Na ovaj način, naš kod za crtanje će se direktno referisati na virutelne koordinate, ali će se pravilno prikazati:

```
procedure TForm2.FormPaint(Sender: T0bject);
begin
  SetWindowOrgEx (Canvas.Handle,
    HorzScrollBar.Position,
    VertScrollBar.Position, nil);

  // draw a yellow line
  Canvas.Pen.Width := 30;
  Canvas.Pen.Color := clYellow;
  Canvas.MoveTo (30, 30);
  Canvas.LineTo (1970, 1970);

// and so on ...
...
```

Ovo je verzija programa koju ćete pronaći u izvornom kodu koji ste preuzeli. Pokušajte da koristite program kada se izbaci poziv `SetWindowOrgEx`, da biste videli šta se dešava kada se ne koriste virtuelne koordinate. Videćete da izlaz programa nije korektan — neće moći da skroluje i ista slika će uvek ostati na istoj poziciji bez obzira na operacije skrolovanja.



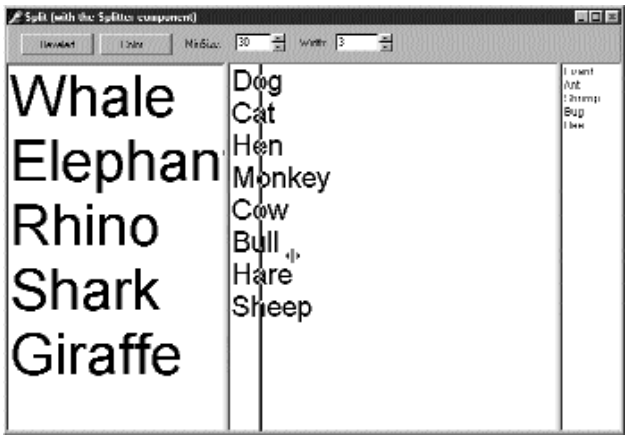
SLIKA 7.14 *Linije koje treba iscrtati na virtuelnoj površini formulara*

Tehnike deljenja formulara

Postoji nekoliko načina na koje možete da implementirate tehnike deljenja formulara u Delphiju, ali najjednostavnji način je upotreba komponente Splitter, koju možete pronaći na strani Additional u okviru Component Palette. Da biste je učinili efikasnijom, Splitter se može koristiti u kombinaciji sa svojstvom `Constraints` kontrola na koje se odnosi. Kao što ćemo videti u primeru Split1, to nam omogućava da definišemo maksimalne i minimalne pozicije Splittera i formulara.

Da biste izradili ovaj primer, jednostavno postavite komponentu ListBox na formular; zatim dodajte komponentu Splitter, još jedan ListBox, još jedan Splitter i na kraju treću komponentu ListBox. Formular takođe sadrži jednostavnu paletu alata zasnovanu na panelu.

Jednostavnim smeštanjem dve Splitter komponente, Vašem formularu dajete potpunu funkcionalnost pomeranja i promene veličine kontrola koje sadrži u vreme izvršavanja. Svojstva `Width`, `Beveld` i `Color` komponenata Splitter određuju njihovo prikazivanje, a u primeru Split1 možete koristiti kontrole palete alata da biste ih izmenili. Još jedno važno svojstvo je svojstvo `MinSize` koje određuje minimalnu veličinu komponenata formulara. Tokom operacije deljenja (videti sliku 7.15) linija obeležava konačnu pozciju Splittera, ali ovu liniju ne možete prevući preko određene granice. Ponašanje programa Split1 ne dozvoljava kontrolama da postanu premale. Alternativna tehnika je određivanje vrednosti `True` za svojstvo `AutoSnap` Splittera. Ovo svojstvo će učiniti da Splitter sakrije kontrolu kada njegova veličina bude manja od granice navedene u svojstvu `MinSize`.



SLIKA 7.15 *Komponeta Splitter primera Split1 određuje minimalnu veličinu za svaku kontrolu formulara, čak i za one koje se ne graniče sa komponentom Splitter*

Savetujem Vam da isprobate program Split1 tako da u potpunosti razumete kako komponenta Splitter utiče na kontrole koje se sa njom graniče i kako utiče na druge kontrole formulara.

Čak i da sam odredio svojstvo `MinSize`, korisnik ovog programa može da smanji veličinu celog formulara na minimum skrivajući time neke liste. Ukoliko testirate verziju primera Split2, imaćete bolje ponašanje. U verziji Split2 sam odredio neka ograničenja (`Constraints`) za kontrole ListBox:

```
object ListBox1: TListBox
  Constraints.MaxHeight = 400
  Constraints.MinHeight = 200
  Constraints.MinWidth = 150
```

Ograničenja veličine se primenjuju onda kada menjate veličinu kontrola, a da bi program zadovoljavajuće funkcionisao, potrebno je da odredite vrednost `rsUpdate` za svojstvo `ResizeStyle`. Ova vrednost označava da se pozicije kontrola ažuriraju za svako pomeranje komponente Splitter, a ne samo na kraju operacije. Ukoliko odaberete vrednost `rsLine` ili novu vrednost `rsPattern`, komponenta Splitter će jednostavno iscrtati liniju na zahtevanoj poziciji proveravajući svojstvo `MinSize` bez ograničavanja kontrola.

SAVET

Komponenta Splitter u Delphiju 5 sadrži novo svojstvo `AutoSnap`. Kada za ovo svojstvo odredite vrednost `True`, Splitter će potpuno sakriti susednu kontrolu kada veličina te kontrole bude manja od minimalne veličine određene za komponentu Splitter. ■

Horizontalna podela

Komponenta Splitter se takođe može upotrebiti za horizontalnu podelu umesto unapred određene vertikalne podele. Ipak, ovaj pristup je nešto komplikovaniji. U osnovi, Vi možete da smestite komponentu na formular, poravnate je uz vrh formulara i zatim smestite Splitter na formular. Unapred je određeno da će biti levo poravnat. Odaberite vrednost `lTop` za svojstvo

`Align` i zatim ručno promenite veličinu komponente tako što ćete promeniti svojstvo Height u Object Inspectoru (ili promenom veličine komponente).

Formular sa horizontalnim Splitterom možete videti u primeru SplitH. Ovaj progam sadrži dve Memo komponente, u kojima možete otvoriti fajl, i sadrži Splitter koji ih odvaja, definisan kao:

```
object Splitter1: TSplirter
  Cursor = crVSplit
  Align = alTop
  OnMoved = Splitter1Moved
end
```

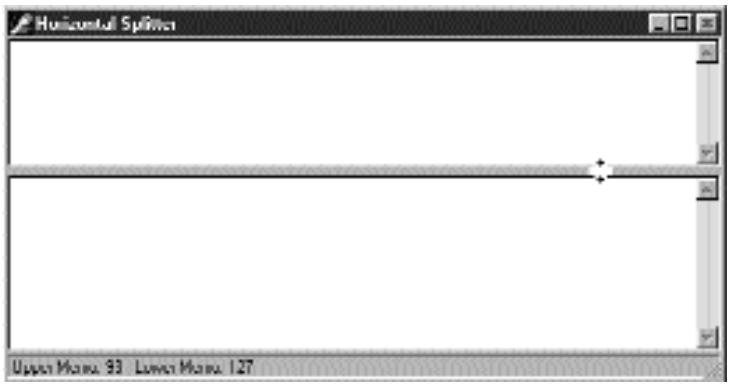
Kada dva puta kliknite, Memo program učitava tekstualni fajl (obratite pažnju na strukturu iskaza `with`):

```
procedure TForm1.MemoDblClick (Sender: TObject);
begin
  with Sender as TMemo, OpenDialog1 do
    if Execute then
      Lines.LoadFromFile (Filename);
end;
```

Program prikazuje statusnu liniju koja prati trenutnu visinu dve Memo komponente. Program obrađuje događaj `OnMoved` Splittera (jedini događaj ove komponente) da bi ažurirao tekst statusne linije. Isti kod se izvršava svaki put kada se promeni veličina formulara:

```
procedure TForm1.SplitterMoved (Sender: TObject);
begin
  StatusBar1.Panels[0].Text : Format (
    'Upper Memo: %d - Lower Memo: %d',
    [MemoUp.Height, MemoDown.Height]);
end;
```

Efekat ovog koda možete videti ukoliko pogledate sliku 7.16, ili pokretanjem primera SplitH.



SLIKA 7.16 *Statusna linija primera SplitH označava poziciju horizontalne komponente Splitter*

Deljenje upotrebom hedera

Alterantiva upotrebi Splittera je upotreba standardne komponente HeaderControl. Ukoliko je postavite na formular, ona će automatski biti poravnata sa vrhom formulara. Zatim možete dodati tri liste ostatku klijent oblasti formulara. Prva lista se može levo poravnati, ali nećete moći da poravnate drugu i treću listu. Problem je u tome što se sekcije hedera mogu prevući izvan vidljive površine formulara. Ukoliko liste koriste automatsko poravnanje, ne mogu se pomeriti van vidljive površine formulara kako to program zahteva.

Rešenje je u definisanju sekcije hedera upotrebom specifičnog editora svojstva `Sections`. Ovo svojstvo editora Vam omogućava da pristupite različitim podobjektima kolekcije i izmenite razne vrednosti. Možete da odredite zaglavlje i poravnanje teksta, trenutnu, minimalnu i maksimalnu veličinu hedera i tako dalje. Određivanje vrednosti ograničenja je moćan alat i zamenjuje svojstvo `MinSize` komponente Splitter ili ograničenja lista koje smo koristili u prethodnim primerima. Izlaz ovog programa, koji je nazvan HdrSplit, možete videti na slici 7.17.

Potrebno je da obradimo dva događaja: `OnSectionResize` i `OnSectionClick`. Prvom obradom se jednostavno menja veličina liste povezane sa modifikovanom selekcijom (koja je određena odgovarajućim brojem svojstvom `ImageIndex` svake od sekcija, a koji se koristi za određivanje naziva liste).

```
procedure TForm1.HeaderControl1SelectionResize (
  HeaderControl: THeaderControl; Section: THeaderSection);
var
  List: TListBox;
begin
  List := FindComponent ('ListBox' + IntToStr (
    Section.ImageIndex)) as TListBox;
  List.Width := Section.Width;
end;
```

SLIKA 7.17 *Izlaz primera HdrSplit*

Pored ovog događaja potrebno je da obradimo promenu veličine formulara koju koristimo za sinhronizovanje lista sa sekcijama kojima se po definiciji menja veličina:

```
procedure TForm1.FormResize(Sender: TObject);
var
  I: Integer;
  List: TListBox;
begin
  for I := 0 to 2 do
  begin
    List := FindComponent ('ListBox' + IntToStr (
      HeaderControl1.Sections[I] .ImageIndex)) as TListBox;
    List.Left := HeaderControl1.Sections[I].Left;
    List.Width := HeaderControl1.Sections[I].Width;
  end;
end;
```

Posle određivanja visine listi ovaj metod jednostavno poziva prethodni prosleđujući mu parametar koji nećemo koristiti u ovom primeru. Drugi metod za HeaderControl, koji se poziva kao odgovor na klik mišem na jednu od sekcija, se koristi za sortiranje sadržaja odgovarajuće liste:

```
procedure TForm1.HeaderControl1SectionClick(
  HeaderControl: THeaderControl; Section: THeaderSection);
var
  List: TListBox;
begin
  List := FindComponent ('ListBox' + IntToStr (
    Section.ImageIndex)) as TListBox;
  List.Sorted := not List.Sorted;
end;
```

Naravno, ovaj kod ne obezbeđuje uobičajeno ponašanje sortiranja elemenata kada kliknete heder i ne obezbeđuje sortiranje u obrnutom redosledu kada ponovo kliknete heder. Da biste to implementirali, potrebno je da napišete sopstveni algoritam sortiranja. Konačno, primer HdrSplit koristi novu karakteristiku kontrole hedera. Primer određuje vrednost svojstva `DragReorder` da bi se omogućile operacije prevlačenja za promenu redosleda sekcija hedera. Kada se obavi ova operacija, kontrola inicira događaj `OnSectionDrag` u kojem možete zameniti pozicije lista. Ovaj događaj se inicira pre nego što se zapravo pomere sekcije, te sam morao da koristim koordinate ostalih sekcija:

```
procedure TForm1.HeaderControl1SectionDrag (Sender: TObject;
FromSection,
  ToSection: THeaderSection; var AllowDrag: Boolean);
var
  List: TListBox;
begin
  List := FindComponent ('ListBox' + IntToStr (
    FromSection.ImageIndex)) as TListBox;
  List.Left := ToSection.Left;
  ListWidth := ToSection.Width;

  List := FindComponent ('ListBox + IntToStr (
    ToSection.ImageIndex)) as TListBox:
  List.Left := Fromsection.Left;
  List.Width := FromSection.Width;
end;
```

Sidra kontrola

U ovom poglavlju sam opisao kako možete da upotrebite poravnanje i Splittere za kreiranje lepih i fleksibilnih korisničkih interfejsa koji se prilagođavaju trenutnoj veličini formulara, dajući korisnicima maksimalnu slobodu. Delphi, takođe, podržava desno i donje usidrenje. Pre nego što je ova karakteristika bila predstavljena u Delphiju, svaka kontrola postavljena na formular je imala koordinate relativne u odnosu na gornje i donje ivice, izuzev ukoliko nije bila poravnata sa gornjom ili donjom ivicom. Za neke kontrole poravnanje je dobro, ali to nije slučaj sa svim kontrolama.

Upotrebom sidra poziciju kontrole možete učiniti relativnom u odnosu na bilo koju stranu formulara. Na primer, da bi kontrola bila usidrena uz donji desni ugao formulara, postavite kontrolu na željenu poziciju i za svojstvo `Anchors` odredite vrednost `[akRight, akBottom]`. Kada se promeni veličina formulara, rastojanje kontrole od mesta sidra se održava stalnim. Drugim rečima, ukoliko odredite ova dva sidra i uklonite dva unapred određena, kontrola će ostati u donjem desnom uglu.

S druge strane, ukoliko postavite velike komponente, kao što su Memo ili ListBox, na sredinu formulara, možete odrediti svojstvo `Anchors` tako da uključuje sve četiri strane. Na ovaj način kontrola će se ponašati kao poravnata kontrola koja će rasti i smanjivati se prema veličini formulara, ali će postojati nekakvo rastojanje između kontrole i strana formulara.

SAVET

Sidra, kao i ograničenja, funkcionišu u vreme dizajniranja i u vreme izvršavanja, te bi trebalo da ih odredite što je pre moguće da biste izvukli korist iz ove karakteristike prilikom dizajniranja formulara kao i u vreme izvršavanja. ■

Kao primer oba pristupa možete isprobati aplikaciju Anchors koja sadrži dve kontrole u donjem desnom uglu i listu na sredini. Kao što je prikazano na slici 7.18, kontrole se automatski pomeraju i menjaju veličinu u skladu sa promenom veličine formulara. Da bi ovaj formular pravilno funkcionisao, morate odrediti i svojstvo `Constraints` inače, kada formular postane premali, kontrole će se preklapati ili nestati.

SLIKA 7.18 *Kontrole primera Anchors se pomeraju i menjaju veličinu onako kako korisnik menja veličinu formulara. Nije potreban nikakav kod za pomeranje kontrola već samo pravilna upotreba svojstva Anchors.*

SAVET

Ukoliko uklonite sva sidra, ili dva naspramna sidra (na primer, levo i desno), operacije promene veličine će učiniti da kontrola bude pokretna. Kontrola će zadržati veličinu, a sistem će dodavati ili uklanjati jednak broj piksela na svakoj strani. Ovo se može definisati kao centralno sidro, jer ukoliko je komponenta inicijalno na sredini formulara, ona će zadržati tu poziciju. U svakom slučaju, ukoliko želite centriranu kontrolu, trebalo bi da koristite naspramna sidra tako da se, ukoliko korisnik poveća formular, poveća i veličina kontrole. U slučaju koji je prikazan, povećanje formulara ostavlja malu kontrolu u centru. ■

Dokiranje paleta alata i kontrola

Još jedna karakteristika koja je dodata u Delphiju 4 bila je podrška za *palete* alata i kontrole koje se mogu dokirati. Drugim rečima, Vi možete kreirati paletu alata i pomeriti je na bilo koju stranu formulara, ili je čak slobodno pomerati po ekranu kada nije dokirana. Ipak, pravilno podešavanje programa, da bi se postigao ovakav efekat, nije lako kao što se čini.

Prvo, Delphi podrška za dokiranje je povezana sa kontejnerskim kontrolama, a ne sa formularima. Panel, ControlBar i ostali kontejneri (dakle, bilo koja kontrola koja je izvedena iz klase `TWinControl`) mogu biti podešeni za dokiranje uključivanjem svojstva `DockSite`. Takođe, možete odrediti svojstvo `AutoSize` ovih kontejnera tako da se prikazuju samo ukoliko sadrže kontrolu.

Da biste mogli da prevlačite kontrolu (objekat bilo koje klase izvedene iz `TControl`) na mesto dokiranja, jednostavno odredite vrednost `dkDock` za njeno svojstvo `DragKind` i vrednost `dmAutomatic` za svojstvo `DragMode`. Na ovaj način kontrola se može prevući sa njene trenutne pozicije na novi kontejner koji se može dokirati. Da kontrola više ne bude dokirana i da možete da je prevučete na specijalni formular, možete da podesite svojstvo `FloatingDockSiteClass` u `TCustomDockForm` (da biste koristili unapred određen samostalni formular koji sadrži sopstveno malo zaglavlje).

Sve operacije dokiranja i uklanjanja dokiranja se mogu pratiti upotrebom specijalnih događaja komponente koja se prevlači (`OnStartDock` i `OnEndDock`) i komponente koja prihvata kontrolu koja je dokirana (`OnDragOver` i `OnDragDrop`). Ovi događaji dokiranja su veoma slični događajima prevlačenja koji su na raspolaganju u prethodnim verzijama Delphija.

Takođe, postoje komande koje možete upotrebiti da biste ostvarili operacije dokiranja u okviru koda i da biste istražili status kontejnera koji je dokiran. Svaka kontrola se može pomeriti na drugu poziciju upotrebom metoda `Dock`, `ManualDock` i `ManualFloat`. Kontejner sadrži svojstvo `DockClientCount`, koje označava broj dokiranih kontrola, i svojstvo `DockClients`, u kojem se nalazi niz ovih kontrola.

Ukoliko je za dokirani kontejner određena vrednost `True`, za svojstvo `UseDockManager` moći ćete da koristite svojstvo `DockManager`, koje implementira interfejs `IDockManager`. Ovaj interfejs sadrži mnoge karakteristike koje možete upotrebiti da biste prilagodili ponašanje kontejnera koji se dokira, uključujući čak i podršku za usmeravanje statusa.

Kao što možete videti iz ovog kratkog opisa, podrška dokiranju u Delphiju se bazira na velikom broju svojstava, događaja, metoda i objekata (kao što su zone dokiranja i stabla dokiranja) — mnogo više karakteristika nego što mi imamo mesta za detaljno opisivanje. Sledeći primer predstavlja glavne karakteristike koje su Vam obično potrebne.

Dokiranje paleta alata u ControlBarovima

Primer MdEdit je konačna verzija editora RichEdit koji je predstavljen u ovom poglavlju. Ova nova verzija sadrži drugi ControlBar u dnu formulara na koji možete prevući jednu od paleta alata koje se nalaze u gornjem ControlBaru. Kako je za oba kontejnera palete alata određena vrednost `True` za svojstvo `AutoSize`, obe palete alata se uklanjaju kada ne sadrže kontrole. Da biste omogućili korisnicima da prevlače palete alata istim sidrom koje se koristi za njihovo pomeranje unutar kontejnera, ne zaboravite da podesite svojstvo `AutoDrag` za ControlBarove.

Primer programa u vreme izvršavanja možete videti na slici 7.19. Za komponente unutar ControlBara na vrhu određena je vrednost `dkDock` za svojstvo `DragKind`. Ipak, meni paleta alata se ne može pomeriti van kontejnera jer želimo da ga zadržimo na tipičnoj poziciji linije menija. Combo polje se može prevući, ali ne želimo da ga korisnik dokira na donjem ControlBaru. Implementiraćemo drugo ograničenje u obradi događaja `OnDockOver` prihvatanjem dokiranja samo za palete alata:

```
procedure TFormRichNote.ControlBarLowerDockOver (Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
begin
  Accept := Source.Control is Toolbar;
end;
```

SLIKA 7.19 *Primer MdEdit6 Vam omogućava da dokirate palete alata (i meni) na vrhu ili dnu formulara ili da ih ostavite pokretnim*

Zatim, želimo da imamo borduru za donji ControlBar, ali samo kada sadrži komponente tako da se ne prikazuje prazna bordura (jer se ControlBar prikazuje kao veoma tanka linija kada je prazan). Da bismo ovo postigli, možemo da dodamo borduru svaki put kada se kontrola smešta na ControlBar (`OnDockDrop`), i da uklonimo borduru kada se poslednja kontrola ukloni (`OnUnDock`). Da bismo odredili broj kontrola, možemo upotrebiti svojstvo `DockClientCount` koje se ažurira pošto se završi uklanjanje dokiranja, te je njegova vrednost još uvek 1 dok se za poslednju kontrolu uklanja dokiranje:

```
procedure TFormRichNote.ControlBarLowerDockDrop (Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
```

```
begin
  ControlBarLower.BevelKind := bkTile;
end;

procedure TFormRichNote.ControlBarLowerUnDock (Sender: TObject;
  Client:  TControl; NewTarget: TWinControl; var Allow: Boolean);
begin
  if ControlBarLower.DockClientCount = 1 then
    ControlBarLower.BevelKind := bkNone;
end;
```

Ovaj isečak iz DFM fajla prikazuje svojstva koja imaju veze sa podrškom za dokiranje:

```
object FormRi chNote: TFormRi chNote
  object RichEdit: TRichEdit . . .
  object ControlBar: TControlBar
    AutoSize := True
    object ToolBarFile: TToolBar
      DragKind = dkDock
      DragMode = dmAutomatic
    end
    object ToolBarEdit: TToolBar
      DragKind = dkDock
      DragMode = dmAutomatic
    end
    object ToolBarFont: TToolBar
      DragKind = dkDock
      DragMode = dmAutomatic
    end
    object ComboFont: TComboBox
      DragKind = dkDock
      DragMode = dmAutomatic
    end
    object ToolBarMenu: TToolBar ...
  end
  object StatusBar: TStatusBar...
  object ControlBarLower: TControlBar
    BevelKind = bkNone
    OnDockDrop = ControlBarLowerDockDrop
    OnDockOver = ControlBarLowerDockOver
    OnUnDock = ControlBarLowerUnDock
  end ...
end
```

UPOZORENJE

Kada pomerite jednu od paleta alata na automatski kreiran pokretni formular, možda ćete doći u iskušenje da ga vratite nazad i zatvorite pokretni formular. Ovo neće funkcionisati jer se pokretni formular uklanja zajedno sa paletom alata koju sadrži. Ipak, možete upotrebiti iskačući meni gornjeg ControlBara da biste prikazali ovu sakrivenu paletu alata. ■

Kontrolisanje operacija dokiranja

Delphi obezbeđuje mnoge događaje i metode koji Vam daju dosta kontrole nad operacijama dokiranja, uključujući i administraciju dokiranja. Da biste se upoznali sa nekim od ovih karakteristika, isprobajte primer DockTest. Program pridružuje svojstvo `FloatingDockSiteClass` komponente Memo za `TForm2` tako da možete da dizajnirate specifične karakteristike i dodate ih pokretnom okviru koji će sadržati kontrolu kada je ona pokretna, umesto da koristite unapred određenu instancu klase `TCustomDockForm`.

Druga karakteristika programa je da obrađuje događaje `OnDockOver` i `OnDockDrop` panela za dokiranje da bi se korisniku prikazale poruke, kao što je poruka o broju kontrola koje su trenutno dokirane:

```
procedure TForm1.Panel1DockDrop (Sender: TObject;
  Source: TDragDockObject; X, Y,: Integer);
begin
  Caption := 'Docked ' + IntTostr (
    Panel1.DockClientCount);
end;
```

Na isti način program obrađuje događaje dokiranja glavnog formulara. Još jedna kontrola, lista, sadrži iskačući meni koji možete pozvati da biste obavili dokiranje i uklonili dokiranje u okviru koda bez upotebe uobičajenog prevlačenja mišem:

```
procedure TForm1.DocktoPanel1Click(Sender TObject);
begin
  // dock to the panel
  ListBoxl.ManualDock (Panel1, Panel1, alBottom);
end;

procedure TForml.DocktoForm1Click(Sender TObject);
begin
  // dock to the current form
  ListBoxl.Dock (Self, Rect (200, 100, 100, 100));
end;

procedure TForm1.Floating1Click(Sender: TObject);
begin
  // toggle the floating status
  if ListBox1.Floating then
    ListBox1.ManualDock (Panell, Panell, alBottom)
  else
    ListBox1.ManualFloat (Rect (100, 100, 200, 300));
  Floating1.Checked := ListBox1.Floating;
end;
```

Poslednja karakteristika ovog primera je verovatno i najinteresantnija. Svaki put kada program prestane da se izvršava, on čuva trenutni status dokiranja panela. Kada se program ponovo pokrene, ponovo primenjuje informacije dokiranja, restaurirajući prethodnu konfiguraciju prozora. Program ovo obavlja samo za panel pa će ostali pokretni prozori biti prikazani na njihovim prvobitnim pozicijama. Evo koda za čuvanje i učitavanje:

```
procedure TForm1.FormDestroy(Sender: TObject);
var
  FileStr: TFileStream;
begin
  if Panel1.DockClientCount > 0 then
  begin
    FileStr := TFileStream.Create (DockFileName,
      fmCreate or fmOpenWrite);
    try
      Panel1.DockManager.SaveToStream (FileStr);
    finally
      FileStr.Free;
    end;
  end
  else
    // remove the file
    DeleteFile (DockFileName);
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  FileStr: TFileStream;
begin
  // reload the settings
  DockFileName := ExtractFilePath (Application.Exename) +
    'dock.dck');
  if FileExists (DockFileName) then
  begin
    FileStr := TFileStream.Create (DockFileName, fmOpenRead);
    try
      Panel1.DockManager.LoadFromStream (FileStr);
    finally
      FileStr.Free;
    end;
  end;
  Panel1.DockManager.ResetBounds (True);
end;
```

Postoji još mnogo karakteristika koje možete testirati, međutim program DockTest već isprobava previše njih. Na primer, automatsko poravnanje ne funkcioniše naročito dobro uz kod dokiranja koji služi za restauraciju. Predlažem da isprobate ponašanje ovog programa proširujući njegovu podršku za tip interfejsa koji Vam se više dopada.

NAPOMENA

Imajte na umu da će, iako dokiranje panela čini aplikaciju lepšom, neki korisnici biti zbunjeni činjenicom da njihove palete alata mogu nestati ili se mogu nalaziti na nekoj drugoj poziciji. Nemojte previše koristiti karakteristiku dokiranja, inače se neki od neiskusnih korisnika mogu zbuniti. ■

Šta je sledeće?

U ovom poglavlju smo se upoznali sa nizom tema vezanim za palete alata i formulare: definicijom palete alata i statusne linije, načinima skrolovanja, deljenja i prevlačenja formulara. Mada ove teme mogu izgledati kao da nemaju dodirne tačke, ipak se sve one odnose na razvoj savremenog korisničkog interfejsa formulara.

Ovo poglavlje možete posmatrati kao prvi korak ka izradi profesionalne aplikacije. Ostale korake ćemo preduzeti u narednim poglavljima. Ali, Vi već imate dovoljno znanja da svoje programe učinite sličnim najbolje prodavanim Windows aplikacijama, što može biti veoma važno Vašim klijentima. Sada kada su elementi glavnog formulara naših programa pravilno podešeni, možemo razmotriti dodavanje sekundarnih formulara i okvira za dijalog. To je tema narednog poglavlja, mada smo već videli koliko je jednostavno dodati drugi formular programu. U narednom poglavlju ćemo se takođe pozabaviti formularima sa više strana, još jednim vrednim dodatkom skupu alata svakog programera koji želi da kreira savremeni korisnički interfejs.

POGLAVLJE

8

Upotreba različitih formulara

SVE DO SADA VEĆINA PROGRAMA OVE KNJIGE JE SADRŽALA SAMO JEDAN FORMULAR. OBIČNO APLIKACIJE SADRŽE GLAVNI PROGRAM, NEKOLIKO POKRETNIH PALETA ALATA ILI PALETA I VELIKI BROJ OKVIRA ZA DIJALOG KOJI SE MOGU POZVATI KOMANDAMA MENIJA ILI KOMANDNIM KONTROLAMA. SLOŽENIJE APLIKACIJE IMAJU MDI STRUKTURU — OKVIRNI PROZOR KOJI SADRŽI VELIKI BROJ PROZORA UNUTAR SVOJE KLIJENT OBLASTI. PROGRAMIRANJE MDI APLIKACIJA ĆE BITI KRATKO OBJAŠNJENO NA KRAJU OVOG POGLAVLJA, POSLE RAZMATRANJA IZRADE OKVIRA ZA DIJALOG I APLIKACIJA SA VIŠE FORMULARA.

Okviri za dijalog nasuprot formularima

Pre nego što Vam prikažem primere aplikacija sa više formulara ili korisnički definisanih okvira za dijalog, dopustite mi da Vam dam opšti opis ovih alternativa. Kada pišete program, zaista ne postoji velika razlika između okvira za dijalog i drugog formulara, ukoliko izuzmemo borduru, ikone bordure i druge elemente korisničkog interfejsa koje možete da prilagodite.

Ono što korisnike vezuje za okvir za dijalog je koncept prioritetnih prozora (modal window) — prozor koji dobija fokus i mora se zatvoriti pre nego što se korisnik vrati u glavni prozor. Ovo važi za poruke i obično važi za okvire za dijalog. Ipak, možete imati i okvire za dijalog koji nisu prioritetni (ili neprioritetni — modeless). Dakle, ukoliko mislite da su okviri za dijalog samo prioritetni formulari, onda ste na dobrom putu, ali Vaš opis nije precizan. U Delphiju (kao i u Windowsu) možete imati neprioritetne okvire za dijalog i prioritetne formulare. Moramo razmotriti dva različita elementa:

- Bordura formulara i korisnički interfejs formulara određuju da li izgleda kao okvir za dijalog.
- Upotreba dva različita metoda (`Show` ili `ShowModal`) za prikazivanje drugog formulara određuje njegovo ponašanje (neprioritetno ili prioritetno).

Dodavanje drugog formulara programu

Da biste dodali drugi formular aplikaciji, dovoljno je da samo kliknete kontrolu New Form, koja se nalazi na Delphi paleti alata, ili da odaberete komandu menija File➥New. Kao alternativu možete odabrati File➥New, preći na stranu Forms ili Dialogs i odabrati jedan od mogućih šablona formulara ili čarobnjaka formulara.

Ukoliko u projektu imate dva formulara, možete upotrebiti kontrolu Select Form ili Select Unit sa Delphi palete alata da biste prelazili sa jednog na drugi formular u vreme dizajniranja. Takođe, možete odrediti koji formular je glavni formular i koje formulare bi trebalo automatski kreirati prilikom pokretanja aplikacije upotrebom strane Forms okvira za dijalog Project Options. Ova informacija se odslikava na izvorni kod fajla projekta.

SAVET

Sekundarni formulari se automatski kreiraju u fajlu izvornog koda projekta prema novoj opciji Delphija 5, polju za potvrdu Auto Create Forms, koje se nalazi na strani Preferences okvira za dijalog Environment Options. Mada je automatsko kreiranje najjednostavniji i najpouzdaniji pristup za programere početnike i za projekte koje treba brzo završiti, ja Vam predlažem da isključite ovu opciju ukoliko želite ozbiljno programiranje. Kada Vaša aplikacija sadrži stotine formulara, ne bi trebalo sve da ih kreirate prilikom pokretanja programa. Kreirajte instance sekundarnih formulara kada i tamo gde su Vam potrebne, i oslobodite ih se kada Vam nisu potrebne. ■

Kada ste pripremili sekundarni formular, možete odrediti vrednost `True` za svojstvo `Visible` i oba formulara će se prikazati prilikom pokretanja programa. Uopšte, sekundarni formulari aplikacije nisu vidljivi, a prikazuju se pozivom metoda `Show` (ili određivanjem vrednosti `True` svojstva `Visible` u vreme izvršavanja). Ukoliko koristite funkciju `Show`, sekundarni formular će se prikazati kao neprioritetan formular tako da se možete vratiti na prvi formular dok je drugi formular još uvek vidljiv. Da biste zatvorili drugi formular, možete upotrebiti sistemski meni ili kliknuti kontrolu ili element menija koji poziva metod `Close`. Kao što smo videli u Poglavlju 6, unapred određena akcija zatvaranja (pogledajte događaj `Close`) za sekundari formular je da se on samo sakrije, dakle, sekundarni formular se ne uklanja prilikom zatvaranja. Sekundarni formular se čuva u memoriji (ponovimo, to nije najbolji pristup) i na raspolaganju Vam je ukoliko želite da ga ponovo prikažete.

Kreiranje sekundarnih formulara u vreme izvršavanja

Izuzev ukoliko ne kreirate formulare prilikom pokretanja programa, potrebno je da proverite da li formular postoji, i da ga kreirate ukoliko je potrebno. Najjednostavniji slučaj je kada želite da kreirate više kopija jednog formulara u vreme izvršavanja. U primeru MultiWin ja sam to učinio napisavši sledeći kod:

```
procedure TForm1.btnMultipleClick (Sender: TObject);
begin
  with TForm3.Create (Application) do
    Show;
end;
```

Svaki put kada kliknete kontrolu, kreira se nova kopija formulara. Primetite da ja ne koristim globalnu promenljivu `Form3` jer nema mnogo smisla dodeliti ovoj promenljivoj novu vrednost svaki put kada kreirate novi formular objekta. Važno je da se u kodu formulara i u drugim delovima aplikacije ne referišete na globalni objekat `Form3`. Promenljiva `Form3` će, neizostavno, biti pokazivač na vrednost `nil`, te bi trebalo da je uklonite iz jedinice da biste izbegli zabunu.

SAVET

U kodu formulara nikada ne bi trebalo da se eksplicitno referišete na formular upotrebom globalne promenljive koju Delphi pripremi za Vas. Na primer, pretpostavimo da se u kodu za `TForm3` referišete na `Form3.Caption`. Ukoliko kreirate sekundarni objekat istog tipa (klase `TForm3`), izraz `Form3.Caption` će se neizostavno odnositi na zaglavlje objekta formulara referisanog promenljivom `Form3`, što ne mora biti objekat koji trenutno izvršava kod. Da biste izbegli ovaj problem, referišite se na svojstvo `Caption` u metodu formulara da biste naznačili zaglavlje aktuelnog objekta formulara, i upotrebite ključnu reč `Self` kada Vam je potrebna određena referenca na objekat aktuelnog formulara. Da biste izbegli bilo kakav problem prilikom kreiranja više kopija formulara, predlažem Vam da uklonite globalni formular objekta iz interfejs dela jedinice koja deklariše formular. Ova globalna promenljiva je neophodna samo za automatsko kreiranje formulara. ■

Kada dinamički kreirate više kopija formulara, ne zaboravite da uklonite svaki formular objekta kada ga zatvorite, obradom odgovarajućeg događaja:

```
procedure TForm3.FormClose (Sender: TObject;
  var Action: TCloseAction);
begin
  Action := caFree;
end;
```

Ukoliko to ne učinite, rezultat će biti veliko zauzimanje memorije jer će svi formulari koje kreirate (kako prozori tako i Delphi objekti) biti čuvani u memoriji i samo će biti sakriveni.

Obratimo sada pažnju na dinamičko kreiranje formulara u programima koji sadrže samo jednu kopiju formulara u datom trenutku. Kreiranje prioritetnog formulara je prilično jednostavno jer okvir za dijalog može da se ukloni prilikom zatvaranja pomoću koda koji je nalik ovome:

```
procedure TForm1.btnModalClick (Sender: TObject);
var
  Modal: TForm4;
begin
  Modal := TForm4.Create (Application);
  try
    Modal.ShowModal;
  finally
    Modal.Free;
  end;
end;
```

Pošto poziv `ShowModal` dovodi do izuzetka, trebalo bi da ga napišete u `finally` bloku da biste se postarali da će se objekti dealocirati. Obično ovaj blok sadrži i kod kojim se inicijalizuje okvir za dijalog pre nego što se prikaže, i kod koji izdvaja vrednosti koje je odredio korisnik pre uklanjanja formulara. Konačne vrednosti su samo za čitanje ukoliko je rezultat `ShowModal` funkcije `mrOK`, kao što ćemo videti u narednom primeru.

Situacija je nešto komplikovanija kada želite da prikažete samo jednu kopiju neprioritetnog formulara. Zapravo, potrebno je da kreirate formular, ukoliko nije već na raspolaganju, i da ga zatim prikažete:

```
procedure TForm1.btnSingleClick (Sender: TObject);
begin
  if not Assigned (Form2) then
    Form2 := TForm2.Create (Application);
  Form2.Show;
end;
```

Ovim kodom se kreira formular prvi put kada je potreban, a zatim se čuva u memoriji, bilo da je vidljiv na ekranu bilo da je sakriven. Da biste izbegli nepotrebno zauzimanje memorije i sistemskih resursa, potrebno je da uklonite sekundarni formular kada ga zatvorite. To možete učiniti ukoliko napišete obradu za događaj `OnClose`:

```
procedure TForm1.FormClose (Sender: TObject,
  var Action: TCloseAction);
begin
  Action : caFree;
  // important: set pointer to nil!
  Form2 := nil;
end;
```

Primetićete da je posle uklanjanja formulara, globalnoj promenljivoj `Form2` dodeljena vrednost `nil`. Bez ovog dela koda zatvaranje formulara bi uklonilo objekat, ali bi se promenljiva `Form2` još uvek odnosila na originalnu lokaciju u memoriji. Kada biste u ovom trenutku pokušali da još jednom prikažete formular upotrebom metoda `btnSingleClick`, koji ste ranije videli, test `if`

`not Assigned` bi dao vrednost `True` jer on jednostavno proverava da li je vrednost promenljive `Form2` `nil`. Kod ne može da kreira novi objekat, a metod Show, koji je pozvan za nepostojeći objekat, će proizvesti grešku sistemske memorije.

Radi eksperimenta možete da generišete ovu grešku uklanjanjem poslednje linije prethodnog koda. Kao što smo videli, rešenje je da objektu `Form2` dodelimo vrednost `nil` kada se objekat ukloni, tako da će pravilno napisan kod "videti" da je potrebno kreirati novi formular pre upotrebe. Ponoviću, eksperimentisanje sa primerom MultiWin se može pokazati korisnim za testiranje različitih uslova. Ja nisam dao nijednu sliku ekrana ovog primera jer su formulari koje prikazuje prilično prazni (sasvim prazni izuzev glavnog formulara koji sadrži tri kontrole).

NAPOMENA

Dodeljivanje vrednosti `nil` promenljivoj formulara ima smisla i funkcioniše ukoliko treba da postoji samo jedna instanca formulara u bilo kom trenutku. Ukoliko želite da kreirate više kopija formulara, potrebno je da koristite druge tehnike kojima pratite koliko ima formulara. Takođe je potrebno da imate na umu da u tom slučaju ne možete koristiti novu proceduru Delphija 5, proceduru `FreeAndNil`, jer se ne može pozvati Free za `Form2`. Razlog tome je što ne možemo da uklonimo formular pre nego što se završe obrade događaja formulara. ■

SPAJANJE MENIJA FORMULARA

Ovo je još jedna karakteristika neprioritetnih formulara koju vredi pomenuti. Mada svaki formular aplikacije može da sadrži sopstvenu liniju menija, možete, takođe, upotrebiti Delphijevu tehniku spajanja menija da biste elemente menija sekudarnog formulara premestili na liniju menija glavnog formulara. Ova tehnika je naročito korisna kod MDI aplikacija, ali je manje interesantna kada su u pitanju neprioritetni formulari jer takvo ponašanje može da zbuni korisnika.

Kod ove tehnike glavni prozor aplikacije sadrži kao i obično liniju menija. Ostali formulari sadrže liniju menija za koju je uključeno svojstvo `AutoMerge`, tako da njihove linije menija neće biti prikazane u okviru formulara već će biti spojene sa linijom menija glavnog prozora. Evo pravila spajanja menija. Svaki meni sadrži svojstvo `GroupIndex`. Kada su linije menija spojene, meniji se uređuju na sledeći način:

- Ukoliko elementi dve različite linije menija imaju isti `GroupIndex`, uklanjaju se vrednosti originalnog menija.
- Elementi se uređuju u rastućem poretku vrednosti `GroupIndex`.

Kreiranje okvira za dijalog

Ranije sam naglasio da se okvir za dijalog ne razlikuje mnogo od ostalih formulara. Postoji veoma jednostavan trik za izradu okvira za dijalog umesto formulara. Potrebno je da jednostavno odredite vrednost `bsDialog` za svojstvo formulara `BorderStyle`. Ovom jednostavnom izmenom interfejs formulara nalikuje okviru za dijalog, nema sistemske ikone, nema polja Minimize i Maximize, a sistemski meni možete da aktivirate ukoliko kliknete zaglavlje desnim tasterom miša. Naravno, takav formular ima tanku borduru okvira za dijalog koja ne omogućava promenu veličine.

Kada ste načinili formular okvira za dijalog, možete ga prikazati kao prioritetan ili neprioritetan prozor upotrebom uobičajena dva metoda (`Show` i `ShowModal`). Prioritetni okviri za dijalog su mnogo češći nego neprioritetni. To je upravo suprotno od formulara; prioritetne formulare treba izbegavati jer ih korisnici ne očekuju. Sledeća tabela prikazuje kompletnu šemu različitih kombinacija stilova:

Tip prozora	Prioritetan	Neprioritetan
Formular	Nikada ne koristiti	Uobičajeno za SDI aplikacije
Okvir za dijalog	Najčešći tip sekundarnog formulara	Koristi se, ali ne često

Da biste izbegli previše sekundarnih formulara, možete kreirati formulare sa više strana, što ćemo razmotriti kasnije u ovom poglavlju. Alternativni pristup je upotreba MDI formulara, što ćemo, takođe, pokazati kasnije u ovom poglavlju.

Okvir za dijalog primera RefList

U Poglavlju 5 smo se pozabavili primerom RefList, koji koristi kontrolu ListView za prikazivanje referenci na knjige, časopise, web sajtove i drugo. U verziji programa RefList2 ja ću jednostavno dodati okvir za dijalog koji se koristi u dve različite situacije: za dodavanje novih elemenata listi i za izmenu postojećih elemenata. Formular okvira za dijalog možete videti na slici 8.1, a njegov tekstualni opis u narednom listingu (detaljno je prikazan jer sadrži mnoge interesantne karakteristike te Vam sugerišem da pažljivo pročitate kod):

```
object FormItem: TFormItem
  Caption = 'Item'
  Color = clBtnFace
  Position = poScreenCenter
  object Label1: TLabel
    Caption = '&Reference:'
    FocusControl = EditReference
  end
  object EditReference: TEdit. . .
  object Label2: TLabel
    Caption = '&Type:'
    FocusControl = ComboType
  end
  object ComboType: TComboBox
    Style = csDropDownList
    Items.Strings = (
      'Book'
      'CD'
      'Magazine'
      'Mail Address'
      'Web Site')
  end
  object Label3: TLabel
    Caption = '&Author:'
    FocusControl = EditAuthor
  end
  object EditAuthor: TEdit...
  object Label4: TLabel
    Caption = '&Country:'
    FocusControl = EditCountry
```

```
  end
  object EditCountry: TEdit...
  object BitBtn1: TBitBtn
    Kind = bkOK
  end
  object BitBtn2: TBitBtn
    Kind = bkCancel
  end
end
```

SLIKA 8.1 *Formular okvira za dijalog primera RefList2 u vreme dizajniranja*

SAVET

Elementi combo polja ovog okvira za dijalog opisuju dostupne slike liste slika tako da korisnik može da odabere tip elementa, a sistem će prikazati odgovarajuću sliku. Još bolje rešenje bi bilo da se prikažu slike u combo polju zajedno sa njihovim opisom. ■

Kao što sam pomenuo, ovaj okvir za dijalog se koristi u dva različita slučaja. Prvi slučaj je kada korisnik odabere File➥Add Items iz menija:

```
procedure TForm1.AddItems1Click (Sender: TObject);
var
  NewItem: TListItem;
begin
  FormItem.Caption := 'New Item';
  FormItem.Clear;
  if FormItem.ShowModal mrOK then
  begin
    NewItem := ListView1.Items.Add;
    NewItem.Caption := FormItem.EditReference.Text;
    NewItem. ImageIndex = FormItem.ComboType.ItemIndex;
    NewItem.SubItems.Add (FormItem.EditAuthor.Text);
    NewItem.SubItems.Add (FormItem.EditCountry.Text);
  end;
end;
```

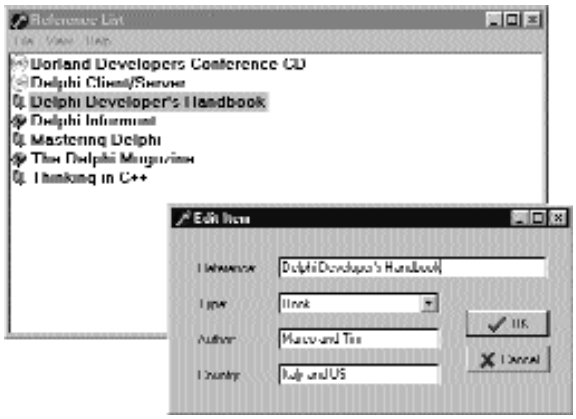
Pored određivanja odgovarajućeg zaglavlja, ova procedura treba da inicijalizuje okvir za dijalog jer mi unosimo potpuno novu vrednost. Ukoliko korisnik klikne OK, program dodaje novi element listi pogleda i određuje sve njene vrednosti. Da bi ispraznio polja za izmene okvira za dijalog, program poziva metod Clear koji uklanja sav tekst svih polja za izmene:

```
procedure TFormItem.Clear;
var
  I: Integer;
begin
  // clear each edit box
  for I := 0 to ControlCount - 1 do
    if Controls [I] is TEdit then
      TEdit (Controls[I]).Text := ''
end;
```

Editovanje postojećeg elementa zahteva malo drugačiji pristup. Prvo, trenutne vrednosti se premeštaju u okvir za dijalog pre nego što se on prikaže. Drugo, ukoliko korisnik klikne OK, program modifikuje trenutnu listu elemenata umesto da kreira novu. Evo koda:

```
procedure TForm1.ListView1DblClick(Sender: TObject);
begin
  if ListView1.Selected <> nil then
  begin
    // dialog initialization
    FormItem.Caption := 'Edit Item';
    FormItem.EditReference.Text := ListView1.Selected.Caption;
    FormItem.ComboType.ItemIndex := ListView1.Selected.ImageIndex;
    FormItem.EditAuthor.Text := ListView1.Selected.SubItems [0];
    FormItem.EditCountry.Text := ListVrew1.Selected.SubItems [1];

    // show it
    if FormItem.ShowModal = mrOK then
    begin
      // read the new values
      ListView1.Selected.Caption := FormItem.EditReference.Text;.
      ListView1.Selected.ImageIndex := FormItem.ComboType.ItemIndex;
      ListView1.Selected.SubItems [0] := FormItem.EditAuthor.Text;
      ListView1.Selected.SubItems [1] := FormItem.EditCountry.Text;
    end;
  end;
end;
```

Efekat ovog koda možete videti na slici 8.2. Primetićete da je kod koji se koristi za čitanje nove vrednosti elementa sličan kodu za modifikovanje. Uopšte uzev, trebalo bi izbegavate ovakvo dupliranje koda i da po mogućstvu smestite zajedničke linije koda u metod koji ćete dodati okviru za dijalog. U ovom slučaju metod bi kao parametar mogao da dobije objekat `TListItem` i da odgovarajuće vrednosti kopira u objekat.



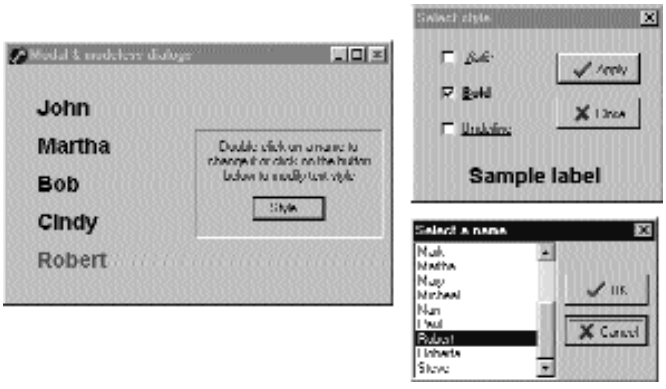
SLIKA 8.2 *Okvir za dijalog primera RefList2 upotrebljen u modu za izmene*

NAPOMENA

Šta se interno dešava kada korisnik klikne kontrole OK ili Cancel okvira za dijalog? Prioritetni okvir za dijalog se zatvara određujući vrednost svojstva `ModalResult` i kao rezultat daje vrednost ovog svojstva. Vrednost rezultata možete naznačiti određivanjem svojstva kontrole `ModalResult`. Kada korisnik klikne kontrolu, vrednost `ModalResult` se kopira u formular, koji zatvara formular i daje vrednost rezultata kao rezultat funkcije `ShowModal`. ■

Neprioritetni okvir za dijalog

Drugi primer okvira za dijalog prikazuje mnogo složeniji prioritetni okvir za dijalog, koji koristi standardni pristup, kao i neprioritetni okvir za dijalog. Glavni formular primera DigApply sadrži pet oznaka sa nazivima kao što možete videti na slici 8.3 i pregledom izvornog koda koji ste preuzeli sa mreže.



SLIKA 8.3 *Tri formulara (glavni formular i tri okvira za dijalog) primera DigApply u vreme izvršavanja*

Ukoliko korisnik klikne naziv, njegova boja prelazi u crvenu; ukoliko korisnik dva puta klikne naziv, program prikazuje prioritetni okvir za dijalog sa spiskom naziva koje možete da odaberete. Ukoliko korisnik klikne kontrolu Style, prikazaće se neprioritetan okvir za dijalog koji korisniku omogućava da promeni stil fonta oznaka glavnog formulara. Pet oznaka glavnog formulara je povezano sa dva metoda; jedan je za događaj `OnClick`, a drugi je za događaj `OnDoubleClick`. Prvi metod pretvara poslednju oznaku koju je korisnik kliknuo u crveno, dok sve ostale oznake prikazuje crnom bojom (čime se za svojstvo Tag određuje vrednost 1, nešto kao indeks grupe). Primetićete da je isti metod povezan sa svim oznakama:

```
procedure TForm1.LabelClick(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to ComponentCount - 1 do
    if (Components[I] is TLabel) and
      (Components[I].Tag = 1) then
    TLabel (Components[Ii).Font.Color := clBlack;
  // set the color of the clicked label to red
  (Sender as TLabel).Font.Color :=  clRed:
end;
```

Drugi metod koji je zajednički za sve oznake je obrada događaja `OnDoubleClick`. Metod `LabelDoubleClick` odabira `Caption` aktuelne oznake (naznačen parametrom `Sender`) iz liste okvira za dijalog, a zatim prikazuje prioritetni okvir za dijalog. Ukoliko korisnik zatvori okvir za dijalog tako što klikne OK i element liste je selektovan, selekcija se kopira nazad u zaglavlje oznake:

```
procedure TForm1.Labe1DoubleC1ick(Sender: TObject);
begin
  with ListDial.ListBox1 do
  begin
    // select the current name in the list box
    ItemIndex := Items.IndexOf (Sender as TLabel).Caption);
    // show the modal dialog box, checking the return value
    if (ListDial.ShowModel = mrOk) and (ItemIndex >= 0) then
      // copy the selected item to the label
      (Sender as TLabel).Caption := Items [ItemIndex];
end;
```

SAVET

Primetićete da se sav kod koji se koristi za prioritetni okvir za dijalog nalazi u metodu `LabelDoubleClick` glavnog formulara. Formular okvira za dijalog ne sadrži dodatni kod. ■

Neprioritetni okvir za dijalog, nasuprot prioritetnom, iza sebe sadrži dosta kodiranja. Glavni formular jednostavno prikazuje okvir za dijalog kada se klikne kontrola Style (primetićete da se zaglavlje kontrole završava trima tačkama da bi se naznačilo da vodi do okvira za dijalog), pozivom metoda `Show`. Okvir za dijalog u vreme izvršavanja možete videti na slici 8.3.

Dve kontrole, Apply i Close, zamenjuju kontrole OK i Cancel u neprioritetnom okviru za dijalog. (Najbrži način da dobijete ove kontrole je da odaberete vrednosti `bkOK` ili `bkCancel` za svojstvo `Kind` i da zatim izmenite `Caption`). Ponekad možete videti da kontrola Cancel funkcioniše kao kontrola Close, ali kontrola OK neprioritetnog okvira za dijalog obično nema značenje. Umesto

toga može postojati jedna ili više kontrola koje izvršavaju određenu akciju glavnog prozora, recimo kao Apply, Change, Style, Replace, Delete i tako dalje.

Ukoliko korisnik klikne jedno od polja za potvrdu ovog neprioritetnog okvira za dijalog, stil teksta oznake na dnu će se promeniti u skladu s tim. Ovo možete postići dodavanjem ili uklanjanjem specifične zastavice koja označava stil, kao što je to učinjeno u narednoj obradi događaja:

```
procedure TStyleDial .ItalicCheckBoxClick(Sender TObject);
begin
  if ItalicCheckBox.Checked then
      LabelSample.Font.Style :=
      LabelSample.Font.Style + [fsItalic]
  else
    LabelSample.Font.Style :=
      LabelSample.Font.Style - [fsItalic]
end;
```

Kada korisnik odabere kontrolu Apply, program kopira stil oznake u svaku oznaku formulara umesto da razmotri vrednosti polja za potvrdu:

```
procedure TStyleDial .ApplyBitBtnClick(Sender; TObject);
begin
  Form1.Label1.Font.Style := LabelSample.Font.Style;
  Form1.Label2.Font.Style := LabelSample.Font.Style;
  ...
```

Kao alternativu, umesto direktong referisanja na svaku oznaku, možete je potražiti pozivanjem metoda formulara `FindComponent`, prosleđujući naziv oznake kao parametar, i zatim konvertovati rezultat u tip `TLabel`. Prednost ovog pristupa je u tome da možemo kreirati nazive raznih oznaka upotrebom petlje `for`:

```
procedure TStyleDial .ApplyBitBtnClick(Sender: TObject);
var
  I: Integer;
begin
  for I := 1 to 5 do
    (Form1.FindComponent ('Label' + IntloStr (I)) as TLabel).
      Font.Style := LabelSample.Font.Style;
end;
```

SAVET

Metod `ApplyBitBtnClick` se takođe može napisati da skenira niz `Controls` u okviru petlje, kao što sam ja to učinio u drugim primerima. Umesto toga, ja sam odlučio da koristim metod `FindComponent` da bih Vam pokazao novu tehniku. ■

Ova druga verzija koda je sasvim sigurno sporija jer mora da obavi više operacija, ali Vi nećete primetiti razliku jer je ionako veoma brza. Naravno, ovaj drugi pristup je mnogo fleksibilniji; ukoliko dodate novu oznaku, potrebno je da samo ispravite gornju granicu petlje `for` jer sve oznake imaju uzastopne brojeve. Primetićete da kada korisnik klikne kontrolu Apply, okvir za dijalog se neće zatvoriti. Samo kontrola Close proizvodi ovaj efekat. Imajte na umu da za ovaj okvir za dijalog nije potreban kod inicijalizacije jer se formular ne uklanja, a njegove komponente zadržavaju svoj status svaki put kada se prikaže okvir za dijalog.

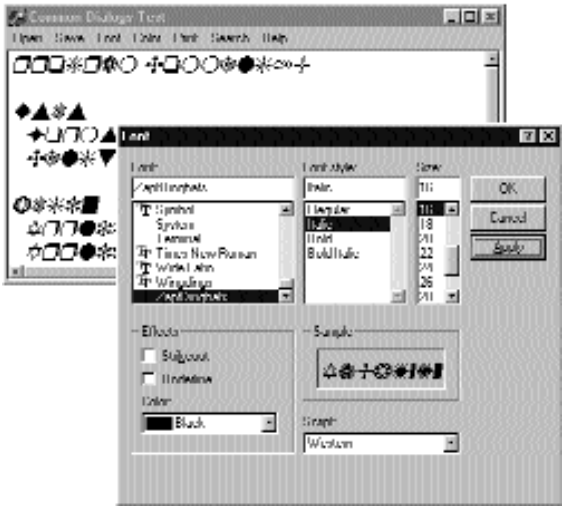
Uobičajeni Windowsovi okviri za dijalog

Pored izrade Vaših okvira za dijalog Delphi Vam omogućava da upotrebite neke od unapred određenih okvira za dijalog različitih tipova. Neki su unapred određeni u Windowsu, a drugi su jednostavni okviri za dijalog (kao što su oni za poruke) koje prikazuje Delphi rutina. Delphi Component Palette sadrži stranu sa komponentama okvira za dijalog. Svaki od ovih okvira za dijalog — poznati su kao uobičajeni Windowsovi okviri za dijalog (Windows common dialogs) — je definisan u sistemskoj biblioteci `ComDlg32.DLL`.

Ja sam već koristio neke od ovih okvira za dijalog u nekoliko primera u prethodnim poglavljima pa ste verovatno sa njima već upoznati. U osnovi, potrebno je da na formular smestite odgovarajuću komponentu, odredite neka svojstva, pokrenete okvir za dijalog (upotrebom metoda `Execute` koji vraća vrednost tipa `Boolean`), i da proverite vrednosti svojstava koja su određena prilikom izvršavanja. Da bih Vam pomogao u eksperimentisanju ovim svojstvima, ja sa napisao test program CommDlg. Neću detaljno razmatrati program niti ću u knjizi prikazati njegov kod. Kao i uvek, kod ovog programa možete pronaći među preuzetim fajlovima.

Ono što želim da uradim je da istaknem neke ključne i ne tako očigledne karakteristike uobičajenih okvira za dijalog, i da Vam omogućim da proučite izvorni kod primera.

- Komponenta Open Dialog se može prilagoditi određivanjem različitih filtara ekstenzija fajlova upotrebom svojstva `Filter` koje sadrži zgodan editor i kojem se direktno može dodeliti string nalik `Text File (*.txt) | *.txt.` Još jedna zgodna karakteristika je da možete prepustiti okviru za dijalog da proveri da li ekstenzija selektovang fajla odgovara unapred određenoj ekstenziji tako što ćete potvrditi zastavicu `ofExtensionDifferent` svojstva `Options` posle izvršavanja okvira za dijalog. Konačno, ovaj okvir za dijalog Vam omogućava da selektujete više elemenata ukoliko podesite opciju `ofAllowMultiSelect`. U tom slučaju spisak selektovanih fajlova možete dobiti pretraživanjem svojstva `Files`.
- Komponenta SaveDialog se koristi na sličan način i sadrži slična svojstva, mada, naravno, ne možete selektovti više od jednog fajla.
- Komponente OpenPictureDialog i SavePictureDialog obezbeđuju slične karakteristike, ali sadrže prilagođen formular koji prikazuje izgled slike. Naravno, ima smisla koristiti ih samo za čuvanje ili otvaranje grafičkih fajlova.
- Komponenta FontDialog se može koristiti za prikazivanje i selektovanje nekog od tipova fontova, fontova koje možete upotrebiti na ekranu i na odabranom štampaču (wysiwyg), ili samo True Type fontova. Možete prikazati deo koji se odnosi na specijalne efekte, ili ga možete sakriti i dobiti i druge različite verzije određivanjem svojstva `Options`. Takođe, možete da aktivirate kontrolu Apply ukoliko obezbedite obradu događaja za događaj `OnApply` i upotrebite opciju `fdApplyButton`. Okvir za dijalog Font sa kontrolom Apply (videti sliku 8.4) se ponaša gotovo kao i neprioritetni okvir za dijalog (mada to nije).
- Komponenta ColorDialog se koristi uz razne opcije da bi se okvir za dijalog u potpunosti prikazao ili ne. Vrednosti svojstva `Options` mogu biti `cdFullOpen` ili `cdPreventFullOpen`.



SLIKA 8.4 *Okvir za dijalog za izbor fontova koji sadrži kontrolu Apply*

- Okviri za dijalog Find i Replace su pravi neprioritetni okviri za dijalog, ali morate sami da implementirate pronalaženje i zamenu, što sam ja delimično učinio u primeru CommDlg. Kod koji obezbeđuje događaje `OnFind` i `OnReplace` je povezan sa kontrolama dva okvira za dijalog.

Parada poruka

Delphi okviri za poruke i okviri za unos predstavljaju još jedan skup unapred određenih okvira za dijalog. Postoji šest Delphi procedura i funkcija koje možete koristiti da biste prikazali jednostavne okvire za dijalog:

- Funkcija `MessageDlg` prikazuje prilagodljiv okvir za poruke, sa jednom ili više kontrola, a obično sadrži i bitmapu. Ovu funkciju smo prilično često koristili u prethodnim primerima.
- Funkcija `MessageDlgPos` je slična funkciji `MessageDlg`. Razlika je u tome što se okvir za poruke prikazuje na zadatoj poziciji, a ne na sredini ekrana.
- Procedura `ShowMessage` prikazuje jednostavnije okvire za poruke sa nazivom aplikacije u zaglavlju i samo sa kontrolom OK. Procedura `ShowMessageFmt` je varijanta procedure `ShowMessage` i sadrži jednak broj parametara kao i funkcija Format. Ova procedura odgovara pozivu funkcije `Format` unutar poziva procedure `ShowMessage`.
- Procedura `ShowMessagePos` ima istu funkciju, samo što se okvir za poruke prikazuje na zadatoj poziciji.
- Metod `MessageBox` objekta `Application` Vam omogućava da odredite i poruku i zaglavlje; možete da obezbedite razne kontrole i karakteristike. Ovo je jednostavna i direktna enkapsulacija Windows API funkcije `MessageBox` koja kao

parametar glavnog prozora prosleđuje obradu objekta `Application`. Ova obrada je neophodna da bi se okvir za poruke ponašao kao prioritetni prozor.

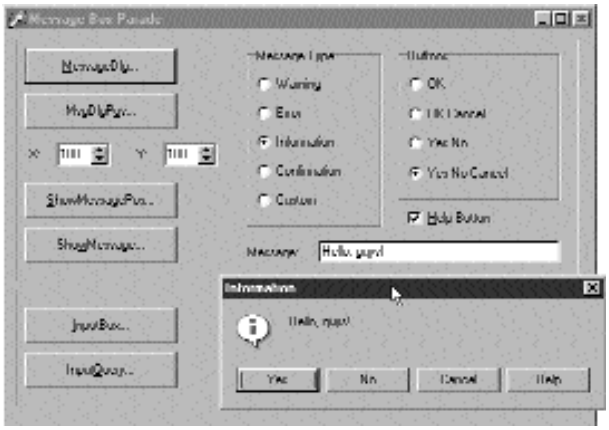
- Funkcija `InputBox` zahteva od korisnika da unese string. Vi obezbeđujete zaglavlje, upit i unapred određeni string.

Funkcija `InputQuery` takođe od korisnika zahteva unos. Jedina razlika između ove funkcije i funkcije `InputBox` je u sintaksi. Funkcija `InputQuery` kao rezultat daje `Boolean` vrednost koja označava da li je korisnik kliknuo OK ili Cancel.

Da bih demonstrirao neke okvire za poruke u Delphiju, ja sam napisao još jedan jednostavan program u kojem koristim sličan pristup kao u prethodnom primeru CommDlg. U primeru MBParade imate veliki broj opcija (opcionih kontrola, polja za potvrdu, polja za izmene i točkića) koje treba podesiti pre nego što kliknete neku od kontrola za prikazivanje okvira za poruke. Bolju predstavu o programu ćete imati ukoliko pogledate formular programa koji je prikazan na slici 8.5.

Prošireni okviri za dijalog

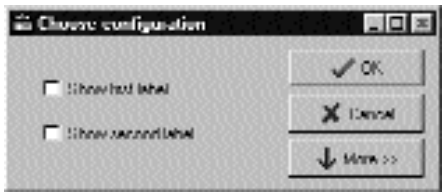
Neki okviri za dijalog prikazuju veliki broj komponenata sa kojima korisnik može da radi. Ponekad možete da podelite okvir za dijalog u logičke celine koje Delphi podržava preko komponente PageControl (razmotrićemo je kasnije u ovom poglavlju), a ponekad možete i da sakrijete neke od kontrola okvira za dijalog da biste pomogli korisnicima koji malo znaju o Vašoj aplikaciji. Alternativa je i povećati okvir za dijalog da bi se prihvatile nove kontrole kada korisnik klikne kontrolu More.



SLIKA 8.5 *Glavni formular primera MBParade sa primerom okvira za poruke*

Ja ću koristiti ovakav pristup pri kreiranju jednostavnog okvira za dijalog u primeru More. Pre svega, potrebno je da kreiramo okvir za dijalog i da dodamo kontrole, kontrolu More (videti sliku 8.6) i dva polja za potvrdu, označena sa *italic* i *bold*, koja se nalaze na panelu van površine formulara u vreme dizajniranja. U praksi, kada jednom dodate panel sa kontrolama, potrebno je da promenite veličinu okvira za dijalog, tako da se novi panel nalazi van vidljive površine

formulara, i da odredite vrednost `False` za svojstvo `AutoScroll`. Panel nije vidljiv jer sam ja uklonio njegovu borduru, a učinio sam program fleksibilnijim jer možete dodati još kontrola sakrivenom delu okvira za dijalog, a da ne morate da menjate izvorni kod: jednostavno ih postavite na panel.



SLIKA 8.6 *Okvir za dijalog primera More u vreme dizajniranja. Neke od komponenata su nevidljive jer se nalaze iza bordure.*

Panel bi trebalo da bude sakriven, inače korisnik može da pritisne taster Tab i pređe na kontrole čak i ako one nisu vidljive. Kao alternativu možete da onemogućite svojstvo `TabStop`. Za ova svojstva (`Visible` ili `TabStop`) se određuje vrednost `True` kada se formular poveća.

Sada je, pored koda koji je neophodan za prenošenje vrednosti iz glavnog formulara u okvir za dijalog, potrebno da napišemo kod kojim se menja veličina formulara kada korisnik klikne kontrolu More. Da bismo pripremili efekat promene veličine, potrebno nam je nekoliko polja na formularu (nazvanih `OldHeight` i `NewHeight`) da bismo sačuvali različite vrednosti klijent oblasti formulara. Vrednosti ovih polja možemo da odredimo prilikom prvog pokretanja formulara:

```
procedure TConfigureDialog.FormCreate (Sender: TObject);
begin
  OldHeight := ClientHeight;
  NewHeight := PanelMore.Top + PanelMore.Height;
end;
```

Ja sam novu visinu dobio dodavanjem visine panela njegovoj poziciji. Promena veličine okvira za dijalog se dešava kada korisnik klikne kontrolu More. Evo prve verzije:

```
procedure TConfigurationDialog.btnMoreClick (Sender: TObject);
begin
  PanelMore.Visible := True;
  btnMore.Enabled := False;
  ClientHeight := NewHeight;
end;
```

Rezultat koji se dobija je prikazan na slici 8.7. Ukoliko želite spektakularniji efekat, možete da povećavate visinu za po piksel umesto da odmah odredite konačnu vrednost. Ukoliko napišete petlju `for` povećavajući visinu klijenta i svaki put ponovo iscrtavajući formular, nove kontrole će se pojavljivati uz lep efekat, ali nešto sporije. Poslednja linija metoda `btnMoreClick` postaje

```
for I := ClientHeight to NewHeight do
begin
  ClientHeight := I;
  Update;
end;
```

Svaki put kada se aktivira okvir za dijalog (događaj `OnFormActivate`), ponovo se određuje njegova visina, sakriva se panel (da bi se izbeglo da korisnik može da upotrebi taster Tab za njegove kontrole) i omogućava se upotreba kontrole More:

```
procedure TConfigureDialog.FormActivate (Sender: TObject);
begin
  Client.Height := OldHeight;
  btnMore.Enabled := True;
  Panel.More.Visible := False;
end;
```

Ovaj kod je neophodan da bi se okvir za dijalog koji se prikazuje, svaki put pokrenuo u unapred određenoj maloj konfiguraciji.

SLIKA 8.7 *Okvir za dijalog primera More pošto mu je promenjena veličina*

Okviri za dijalog About i uvodni ekrani

Windows aplikacije obično sadrže okvir About u kojem se prikazuju informacije kao što su verzija proizvoda, prava o kopiranju i tako dalje. Najjednostavniji način za izradu ovog okvira za dijalog je upotreba funkcije `MessageDlg`. Ovim metodom možete da prikažete samo ograničenu količinu teksta bez specijalne grafike.

Zbog toga je uobičajeni metod za kreiranje okvira About upotreba jednostavnog okvira za dijalog, kao što je onaj koji se generiše pomoću Delphijevih unpared određenih šablona. Kažem *jednostavan* jer kada ste dizajnirali formular sa logotipom i drugim stvarima, bilo Vam je potrebno dosta kodiranja. Neki deo koda može da bude potreban za prikazivanje sistemskih informacija, kao što su verzija Windowsa ili količina slobodne memorije, ili za prikazivanje nekih korisničkih informacija, kao što je registrovano korisničko ime.

NAPOMENA

U Poglavlju 19 ćete videti kako da načinite izvod verzije iz izvršnog fajla koji sadrži ovaj tip Windows resursa. Ova tehnika može da bude korisna pri izradi okvira About koji sadrži informacije o verziji. ■

Izrada korisničkog sakrivenog ekrana

Kada izrađujemo sopstveni okvir About, možemo da dodamo sakriveni ekran, koji imaju Delphi i mnoge druge aplikacije. Možda želite da dodate sakriveni ekran iz više razloga. Ukoliko radite u velikoj kompaniji, to može biti Vaš način na koji hoćete da potvrdite da ste radili na tom projektu, što Vam može pomoći pri pronalaženju novog posla (ukoliko je projekat bio uspešan). Ponekad sakriveni okvir About može biti zabavan, a ponekad ovakvi okviri za dijalog daju dobru priliku da se našalite sa svojim konkurentima. Mnogo ozbiljniji razlog za sakriveni ekran je to što on služi da prikaže ko je napisao program.

Ja sam napisao jednostavan primer koji Vam pokazuje kako biste mogli da implementirate sakriveni okvir za dijalog. Okvir za dijalog sadrži komponentu Panel na kojoj su dve komponente Label. Panel može da sadrži bilo koji broj komponenata da bi se prikazao tekst ili grafika. Neki od stringova se mogu dobiti u vreme izvršavanja. Jedina dodatna karakteristika koja je potrebna za prikazivanje sakrivenog ekrana je komponenta PaintBox koja prekriva deo formulara.

Kada korisnik izvrši određenu složenu akciju (u ovom slučaju kada klikne desnim tasterom miša gornju oznaku dok je pritisnut taster Shift), panel se sakriva i nešto se prikazuje na ekranu. Jednostavno rešenje je ispisati nekakav tekst na formularu:

```
procedure TAboutBox.LabelMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  if (Button = mbRight) and (ssShift in Shift) then
  begin
    Panel1.Visible := False;
    PaintBox1.Canvas.Font.Name := 'Aria 1';
    PaintBox1.Canvas.Font.Size := 20;
    PaintBox1.Canvas.TextOut (40, 50, 'Author: Marco Cantu' );
    PaintBox1.Canvas.TextOut (40, 100, 'Version 1.0');
  end;
end;
```

Da biste izradili spektakularniji sakriveni ekran, možete skrolovati tekst u petlji `for` kao što sam to ja učinio u konačnoj verziji primera Credits. Primetićete da pozicije linija zavise od visine teksta koja se dobija pozivom metoda `TextHeight` za `Canvas` komponente PaintBox:

```
Panell.Visible := False;
LineH := PaintBox1.Canvas.TextHeight ('0');
for I := 0 to 100 +  LineH * 10 do
  with PaintBox1.Canvas do
  begin
    // empty lines are used to delete descendants
    TextOut (40, 100 - I, 'CREDITS example from:');
    TextOut (40, 100 + LineH - I, '''Mastering Delphi''');
    TextOut (40, 100 + LineH * 2 - I, '                          ');

    // wait 5 milliseconds
```

```
    Delay (0, 5);
  end;
Panel1.Visible := True;
```

Da bih izbegao da skrolovanje bude prebrzo, naročito na bržim kompjuterima, ja sam unutar petlje `for` dodao poziv proceduri `Delay` koja kao parametre očekuje sekunde ili milisekunde pauze. Ova procedura `Delay` proverava trenutno vreme, a zatim čeka u petlji `while` sve dok ne prođe zahtevani broj sekundi ili milisekundi:

```
procedure Delay (Seconds, MilliSec: Word);
var
  TimeOut: TDateTime;
begin
  TimeOut := Now + EncodeTime (0,
    Seconds div 60, Seconds mod 60, MilliSec);
 // wait until the TimeOut time
  while Now < TimeOut do
    Application. ProcessMessages;
end;
```

Unutar petlje sam pozvao metod `ProcessMessages` globalnog objekta `Application` da bih Windowsu omogućio da generiše i pošalje potrebnu poruku za iscrtavanje. Ova procedura `Delay` je zaista generička te je lako možete koristiti i u drugim aplikacijama.

SAVET

Uzmite u obzir i drugi aspekt prethodne procedure. Mi smo napisali kod kojim se nešto iscrtava na površini okvira za dijalog. Mada nije uobičajeno, okviri za dijalog mogu da imaju grafički izlaz i mogu da reaguju na akcije mišem baš kao i svaki drugi formular. Zapravo, okvir za dijalog i jeste formular. ■

Izrada uvodnog ekrana

Još jedna tipična tehnika koja se koristi u aplikacijama je prikazivanje inicijalnog ekrana pre nego što se prikaže glavni formular. Ovo aplikaciju čini ugodnijom jer korisniku prikazujete nešto dok se program učitava, ali takođe predstavlja lep vizuelni efekat. Ponekad se isti prozor prikazuje i za okvir About.

Kao primer u kome je uvodni ekran naročito koristan, ja sam izradio program koji prikazuje listu sa prostim brojevima. Prosti brojevi se izračunavaju prilikom pokretanja programa tako da se prikazuju čim formular postane vidljiv:

```
procedure TForm1.FormCreate (Sender: TObject);
var
  I: Integer;
begin
  for I := 1 to 20000 do
    if IsPrime (I) then
      ListBox1.Items.Add (IntToStr (I));
end;
```

Ovaj metod poziva funkciju `ISPrime` koju sam dodao programu. Ova funkcija, koju možete pronaći u izvornom kodu, izračunava proste brojeve veoma sporo; ali, ja sam morao da usporim

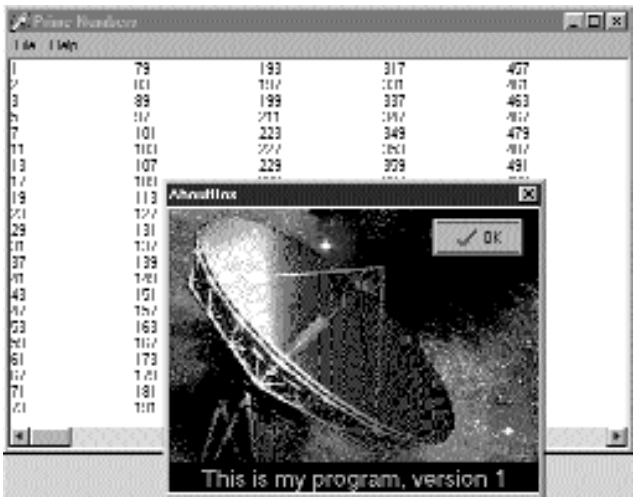
kreiranje formulara da bih Vam pokazao poentu. Brojevi se dodaju u listu koja prekriva celu klijent oblast formulara i omogućava da se prikaže više kolona, kao što možete videti na slici 8.8.

Kao što vidite prilikom pokretanja primera Splash0, problem je u tome što inicijalna operacija, koja se obavlja u metodu `FormCreate`, zahteva dosta vremena. Kada pokrenete program, potrebno je nekoliko sekundi (na standardnoj Pentium mašini) da se prikaže formular. Ukoliko je Vaš kompjuter veoma brz ili veoma spor, možete da promenite gornju granicu petlje `for` metoda `FormCreate` da biste program učinili bržim ili sporijim.

Ovaj program sadrži jednostavan okvir za dijalog sa slikom, jednostavno zaglavlje, bitmapiranu kontrolu, i svi su smešteni unutar panela i zuzimaju celu površinu okvira About. Ovaj formular se prikazuje kada odaberete element menija Help➥About. Ali, ono što zaista želimo da prikažemo je okvir About prilikom pokretanja programa. Ovaj efekat možete da vidite pokretanjem primera Splash1 i Splash2, koji prikazuju uvodni ekran na dva različita načina.

Pre svega, ja sam dodao metod klasi `TAboutBox`. Ovaj metod, nazvan `MakeSplash`, menja neka svojstva formulara da bi formular učinio pogodnim za uvodni ekran. U osnovi, metod uklanja borduru i zaglavlje, sakriva kontrolu OK, borduru panela čini tankom (da bi se zamenila borduра formulara) i zatim prikazuje formular momentalno ga ponovo iscrtavajući (videti sliku 8.9):

```
procedure TAboutBox.MakeSplash
begin
  BorderStyle := bsNone;
  BitBtn1.Visible := False;
  Panel1.BorderWidth := 3;
  Show;
  Update;
end;
```



SLIKA 8.8 *Glavni formular primera Splash, kada je okvir About aktiviran iz menija*

SLIKA 8.9 *Formular uvodnog ekrana primera Splash1 je malo drugačiji od originalnog okvira About (prikazanog na slici 8.8)*

Ovaj metod se poziva posle kreiranja formulara u projekt-fajlu primera Splash1. Ovaj kod se izvršava pre kreiranja ostalih formulara (u ovom slučaju samo glavnog formulara), a zatim se uklanja uvodni ekran pre izvršavanja aplikacije. Ove operacije se obavljaju u bloku `try-finally`. Evo izvornog koda glavnog bloka projekt-fajla za primer Splash2:

```
var
  SplashAbout: TAboutBox;

begin
  Appliacation.Initialize;
  // create and show the splash form
  SplashAbout := TAbouttox.Create (Application);
  try
    SplashAbout.MakeSplash;
    // standard code...
    Application.CreateForm(TForm1, Forml);
    // get rid of the splash form
    SplashAbout.Close;
  finally
    SplashAbout. Free;
  end;

  Application.Run;
end.
```

Ovaj pristup ima smisla samo ukoliko je za glavni formular Vaše aplikacije potrebno nešto vremena za kreiranje, za izvršavanje koda (kao što je to ovde slučaj), ili za otvaranje tabela baze podataka. Primetićete da je uvodni ekran prvi formular koji je kreiran, ali, pošto program ne koristi metod `FormCreate` objekta `Application`, ovaj formular ne postaje glavni formular aplikacije. U tom slučaju, zatvaranje uvodnog ekrana bi prekinulo izvršavanje programa!

Alternativni pristup je nešto duže zadržavanje uvodnog ekrana i upotreba tajmera da bi ga se oslobodili posle nekog vremena. Ja sam ovu drugu tehniku implementirao u primeru Splah2. Ovaj primer takođe koristi drugačiji pristup kreiranju uvodnog ekrana: umesto kreiranja uvodnog ekrana izvornim kodom, primer kreira formular na samom početku metoda `FormCreate` glavnog formulara.

```
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
  SplashAbout: TAboutBox;
begin
  // create and show the splash form
  SplashAbout := TAboutBox.Create (Application);
  SplashAbout. MakeSplash;
  // standard code...
  for I := 1 to 20000 do
    if IsPrime (I) then
      ListBox1.Items.Add (IntToStr (I));
  // get rid of the splash form, after a while
  SplashAbout.Timer1.Enabled := True;
end;
```

NAPOMENA

Ovaj kod se pravilno izvršava bez obzira na redosled kreiranja formulara, kao što je naznačeno svojstvom `OldCreateOrder` (razmatrali smo ovo svojstvo u Poglavlju 6). ■

Tajmer se aktivira pre nego što se završi izvršavanje metoda. Kada protekne određeni period (u primeru je to tri sekunde), aktivira se događaj `OnTimer`, a uvodni ekran ga obrađuje tako što će se zatvoriti i ukloniti formular uvodnog ekrana:

```
procedure TAboutBox.Timer1Timer (Sender: TObject);
begin
  Close;
  Release;
end;
```

NAPOMENA

Metod `Release` formulara je sličan metodu `Free` objekata, jedino što se uklanjanje fomulara odlaže sve dok se ne izvrše sve obrade događaja. Upotreba metoda `Free` može da dovede do narušavanja pristupa jer se interni kod, koji pokreće obradu događaja, može odnositi na formular. ■

Potrebno je popraviti još jednu stvar. Formular Main će biti kasnije prikazan ispred uvodnog ekrana, izuzev ukoliko ne učinite da bude na vrhu. Zbog toga sam dodao jednu liniju metodu `MakeSplash` okvira About u primeru Splash2:

```
FormStyle := fsStayOnTop;
```

Formulari sa više strana

Kada je potrebno da prikažete dosta informacija i kontrola u okviru za dijalog ili formularu, tada možete da koristite više strana. To je metafora beležnice: upotrebom kartica korisnik može odabrati jednu od mogućih strana.

Postoje dve kontrole koje možete upotrebiti za izradu višestranične aplikacije u Delphiju:

- Možete da upotrebite komponentu PageControl Windowsa 95, koja ima kartice na jednoj od strana i više stranica (slično panelima) koje prekrivaju ostatak njene površine. Pošto postoji po jedna strana za svaku karticu, možete da postavite kontrole na svaku stranu da biste dobili efekat, kako u vreme dizajniranja tako i u vreme izvršavanja.
- Možete upotrebiti komponentu TabControl koja ima samo opciju kartica, ali ne nudi stranice koje bi sadržavale informacije. U ovom slučaju ćete koristiti jednu ili više komponenata da biste oponašali operaciju promene stranica.

Treća komponenta koju možete da upotrebite, komponenta TabSheet, predstavlja jednu stranu komponente PageControl. To nije samostalna komponenta i nije dostupna iz Component Palette. Komponentu TabSheet kreirate u vreme dizajniranja upotrebom lokalnog menija komponente PageControl ili u vreme dizajaniranja upotrebom metoda iste kontrole.

NAPOMENA

Delphi još uvek sadrži komponente Notebook, TabSet i TabbedNotebook koje su predstavljene u prethodnim verzijama. Koristite ove komponente samo ukoliko je potrebno da kreirate 16-bitnu verziju aplikacije. Za bilo koju drugu svrhu, komponente PageControl i TabControl, koje enkapsuliraju Win32 kontrole, obezbeđuju daleko savremeniji interfejs. Zapravo, u 32-bitnim verzijama Delphija komponenta TabbedNotebook je ponovo implementirana internom upotrebom Win32 komponente PageControl da bi se reducirala veličina koda i da bi se osavremenio izgled. ■

PageControl i TabSheet

Kao i obično, umesto dupliranja liste svojstava i metoda Help sistema komponente PageControl, ja sam izradio primer koji proširuje njegove mogućnosti i koji Vam omogućava da promenite ponašanje u vreme izvršavanja. Struktura komponente PageControl i drugih ključnih komponenata je ovde prikazana:

```
object Form1: TForm1
  BorderIcons = [biSystemMenu, biMinimize]
  BorderStyle = bsSingle
  Caption = 'Pages Test'
  OnCreate = FormCreate
  object PageControl1: TPageControl
    ActivePage = TabSheet1
    Align = alClient
    HotTrack = True
    Images = ImageList1
    MultiLine = True
    object TabSheet1: TTabSheet
      Caption = 'Pages'
      object Label3: TLabel
      object ListBox1: TListBox
    end
    object TabSheet2: TTabSheet
      Caption = 'Tabs Size'
      ImageIndex = 1
      object Label1: TLabel
      // other controls
```

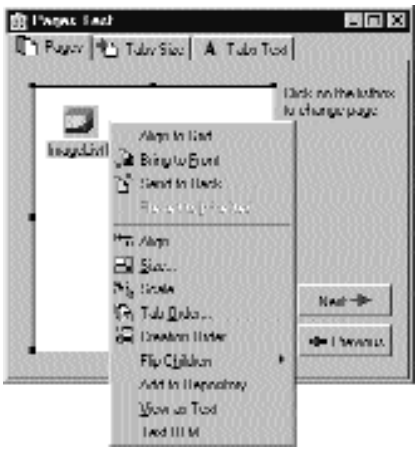
```
    end
    object TabSheet3: TTabSheet
      Caption = 'Tabs Text'
      ImageIndex = 2
      object Memo1: TMemo
        Anchors [akLeft, akTop, akRight, akBottom]
        OnChange = Memo1Change
      end
      object BitBtnChange: TBitBtn
        Anchors = [aklop, akRight]
        Caption = '&Change'
      end
    end
  end
  object BitBtnPrevious: TBitBtn
    Anchors = [akRight, akBottom]
    Caption = '&Previous'
    OnClick = BitBtnPreviousClick
  end
  object BitBtnNext: TBitBtn
    Anchors = [akRight, akBottom]
    Caption = '&Next'
    OnClick = BitBtnNextClick
  end
  object ImageList1: TImageList
    Bitmap = {. ..}
  end
end
```

Primetićete da su kartice povezane sa bitmapama koje obezbeđuje kontrola ImageList i da neke kontrole koriste svojstvo `Anchors` da bi ostale na jednakoj razdaljini od desne ili donje bordure formulara. Čak i ako formular ne podržava promenu veličine (što bi bilo previše komplikovano sa toliko mnogo kontrola), pozicije se mogu promeniti kada se kartice prikažu na više linija (jednostavno povećajte dužinu zaglavlja) ili na levoj strani formulara.

Svaki objekat TabSheet ima svoj `Caption` koji se prikazuje kao kartica. U vreme dizajniranja možete da upotrebite lokalni meni za kreiranje novih strana i za prelazak sa strane na stranu. Lokalni meni komponente PageControl možete videti na slici 8.10, na kojoj je prikazana i prva strana. Ova strana sadrži listu i malo zaglavlje, a sa ostalim stranama deli dve kontrole.

Ukoliko na stranu postavite komponentu, ona je dostupna samo na toj strani. Kako možete da imate istu komponentu (u ovom slučaju dve kontrole) na svakoj strani, a da ih ne duplirate? Jednostavno postavite kontrolu na formular van PageControl (ili pre njnog poravnanja u klijent oblasti), a zatim je pomerite ispred strana, pozivajući komandu Bring to Front iz lokalnog menija formulara. Dve kontrole koje sam ja postavio se mogu koristiti za prelazak sa strane na stranu i predstavljaju alternativu upotrebi kartica. Evo koda ovih kontrola:

```
procedure TForm1.BitBtnNextClick (Sender: TObject);
begin
  PageControl1.SelectNextPage (True);
end;
```



SLIKA 8.10 *Prvi list komponente PageControl primera Pages i njen lokalni meni*

Druga kontrola poziva istu proceduru prosleđujući vrednost `False` kao parametar za selektovanje prethodne strane. Primetićete da nije potrebno proveriti da li se nalazimo na prvoj ili poslednjoj strani jer metod `SelectNextPage` pretpostavlja da je poslednja strana ona koja se nalazi ispred prve i direktno će prelaziti između te dve strane.

Sada ponovo možemo da obratimo pažnju na prvu stranu. Prva strana sadrži listu koja će u vreme izvršavanja prikazivati nazive kartica. Ukoliko korisnik klikne element liste, menja se prikazana strana. Ovo je treći metod koji možete upotrebiti za promenu strana (pored kartica i kontrola Next i Previous). Lista se popunjava u metodu `FormCreate` koji je povezan sa događajem formulara `OnCreate` i kopira zaglavlje svake strane (svojstvo `Page` čuva listu TabSheet objekata):

```
for I := 0 to PageControl1.PageCount - 1 do
  ListBox1.Items.Add (PageControl1.Pages.Caption);
```

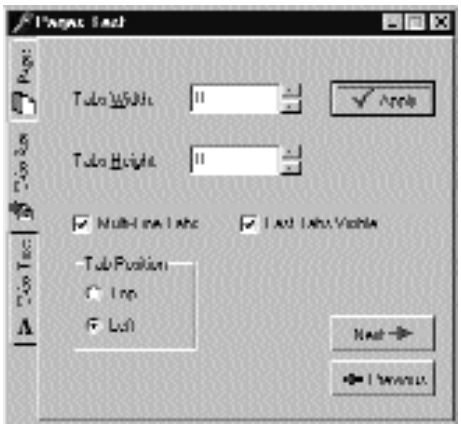
Kada kliknete element liste, možete selektovati odgovarajuću stranu:

```
procedure TForm1.ListBox1Click (Sender: TObject);
begin
  PageControl1.ActivePage :=
    PageControl1.Pages [ListBox1.ItemIndex];
end;
```

Druga strana sadrži dva polja za izmene (koja su povezana sa dve komponente UpDown), dva polja za potvrdu i dve opcione kontrole, što možete videti na slici 8.11. Korisnik može da unese broj (ili odabere broj tako što će kliknuti kontrole naviše ili naniže, ili pritisnuti tastere ↑ ili ↓ kada je odgovarajuće polje za izmene u fokusu), potvrdi polja ili opcione kontrole, a zatim pritisne kontrolu Apply da bi načinio izmene:

```
procedure TForm1.BitBtnAppltClick(Sender: TObject);
begin
  // set tab width, height, and lines
  PageControl1.Tabwidth := StrToInt (EditWidth.Text);
  PageControl1.TabHeight := StrToInt (EditHeight.Text);
  PageControl1.MultiLine := CheckBoxMultiLine.Checked;
```

```
  // show or hide the last tab
  TabSheet3.TabVisible := CheckBoxVisible.Checked;
  // set the tab position
  if RadioButton1.Checked then
    PageControl1.TabPosition := tpTop
  else
    PageControl1.TabPosition := tpLeft;
end;
```



SLIKA 8.11 *Druga strana primera koja se može koristiti za određivanje veličine i pozicije kartica. Na slici su kartice prikazane na levoj strani kontrole.*

Ovim kodom možete da promenite širinu i visinu svake kartice (imajte na umu da 0 znači da se veličina komponente automatski izračunava prema prostoru koji zauzima string), odaberete da imate više linija u kartici ili dve male strelice za skrolovanje oblasti kartice i da ih pomerite na levu stranu. Kontrola, takođe, omogućava da kartice postavite na dno ili na desnu stranu; međutim, naš program to ne dozvoljava jer bi postavljanje ostalih kontrola bilo prilično složeno.

Takođe, možete sakriti poslednju karticu komponente PageControl koja odgovara komponenti `Tabsheet3`. Ukoliko sakrijete jednu od kartica određivanjem vrednosti `False` za svojstvo `TabVisible`, ne možete doći do te kartice tako što ćete kliknuti kontrole Next i Previous, koje funkcionišu na osnovu metoda `SelectNextPage`. Umesto toga bi trebalo da koristite funkciju `FindNextPage`, kao što je prikazano u novoj verziji obrade događaja `OnClick` kontrole Next:

```
procedure TForm1.BitBtnNextClick(Sender: TObject);
begin
  PageControl1.ActivePage :=
    PageControl1. FindNextPage (
      PageControl1.ActivePage, True, False);
end;
```

Poslednja strana sadrži komponentu Memo na kojoj su nazivi strana (dodati su u metodu `FormCreate`). Vi možete da promenite nazive strana i kliknete kontrolu Change da biste izmenili tekst kartica, ali samo ukoliko broj stringova odgovara broju kartica:

```
procedure TForm1.BitBtnChangeClick(Sender: TObject);
var
  I: Integer;
begin
  if Memo1.Lines.Count <> PageControl1.PageCount then
    MessageDlg ('One line per tab, please', mtError, [mbOK], 0)
  else
    for I := 0 to PageControl1.PageCount -1 do
      PageControl1.Pages [I].Caption := Memo1.Lines [I];
  BitBtnChange.Enabled := False;
end;
```

Konačno poslednja kontrola, AddPage, Vam omogućava da dodate novu karticu kontroli, mada na nju program ne dodaje komponente. Objekat tab (prazan) se kreira tako da je PageControl njen vlasnik, ali neće funkcionisati ukoliko ne podesite svojstvo `PageControl`. Pre nego što to učinite, trebalo bi da novu karticu učinite vidljivom. Evo koda:

```
procedure TForm1.BitBtnAddClick(Sender: TObject);
var
  strCaption: string;
  NewTabSheet: TTabSheet;
begin
  strCaption := 'New Tab'
  if InputQuery ('New Tab', 'Tab Caption', strCaption) then
  begin
    // add a new empty page to the control
    NewTabSheet := TTabsheet.Create (PageControl1);
    NewTabSheet.Visible := True;
    NewTabSheet := Caption.strCaption;
    NewTabSheet.PageControl := PageControl1;
    PageControl1.ActivePage = NewTabSheet;
    // add it to both lists
    Memo1.Lines.Add (strCaption);
    ListBox1.Items.Add (strCaption);
  end;
end;
```

SAVET

Kad god napišete formular na osnovu PageControl, ne zaboravite da je prva strana koja se prikaže u vreme izvršavanja strana na kojoj ste bili pre nego što je kod kompajliran. To znači da ćete, ukoliko ste radili na trećoj strani i zatim kompajlirali i pokrenuli program, početi na trećoj strani. Uobičajeni način za rešavanje ovog problema je da dodate liniju koda u metod `FormCreate` da biste PageControl ili Notebook podesili na prvu stranu. Na ovaj način aktuelna strana u vreme dizajniranja ne određuje inicijalnu stranu u vreme izvršavanja. ■

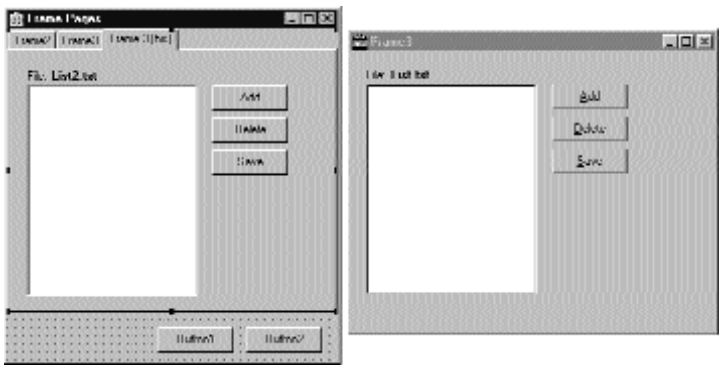
Okviri i strane

Kada imate okvir za dijalog sa mnogo strana prepunih kontrola, kod koji se nalazi iza formulara postaje veoma složen jer su sve kontrole i metodi deklarisani u jednom formularu. Takođe, kreiranje svih ovh komponenata (i njihova inicijalizacija) može da dovede do zastoja pri prikazivanju okvira za dijalog.

Mogućnost upotrebe okvira u Delpjiju 5 (videti prvo i četvrto poglavlje) može rešiti oba problema. Prvo, lako možete da podelite kod jednog složenog formulara u po jedan okvir za svaku stranu. Formular će jednostavno sadržati sve okvire u komponenti PageControl. Ovo Vam sasvim sigurno pomaže da imate jednostavnije i usredsređenije jedinice, i mnogo je lakše ponovo upotrebiti određenu stranu u nekom drugom okviru za dijalog ili aplikaciji. Ponovna upotreba jedne strane komponente PageControl upotrebom okvira ili ugnežđenog formulara je, zapravo, sve sem jednostavna.

Kao primer ovakvog pristupa ja sam izradio primer FramePage koji sadrži nekoliko okvira smeštenih na tri strane komponente PageControl, kao što možete videti na slici 8.12. Svi okviri su poravnati prema klijent oblasti i koriste sav prostor strane na koju su postavljeni.

Zapravo, dve strane sadrže isti okvir, ali dve instance okvira imaju neke razlike u vreme dizajniranja. Okvir, u primeru nazvan `Frame3`, sadrži listu koja se popunjava tekst-fajlom prilikom pokretanja, sadrži kontrole za izmenu elemenata liste i za njihovo čuvanje u fajlu. Naziv fajla je prikazan u oznaci tako da lako možete odabrati fajl za okvir u vreme izvršavanja promenom stringa za `Caption` oznake.



SLIKA 8.12 *Svaka strana primera FramePag sadrži okvir, čime se kod ovog složenog formulara razbija u manje jedinice kojima se lakše rukuje*

SAVET

Mogućnost upotrebe više instanci okvira je jedan od razloga zbog kojih je ova tehnika predstavljena, a prilagođavanje okvira u vreme dizajniranja je još važnije. Pošto dodavanje svojstava okviru i njihova upotreba u vreme dizajniranja zahteva prilagođeni i složen kod, lepo je imati mogućnost upotrebe komponente koja može da prihvati ove vrednosti. Vi imate mogućnost sakrivanja ovih komponenata (recimo, oznake, kao u našem primeru) ukoliko se ne odnose na korisnički interfejs. ■

U primeru je potrebno da učitamo fajl kada se kreira instanca okvira. Pošto okviri nemaju događaj `OnCreate`, verovatno je najbolje zaobići metod `CreateWnd`. Ukoliko napišemo sopstveni konstruktor, nećemo rešiti problem jer će se izvršiti prerano — pre nego što određeni tekst oznake bude dostupan. Evo koda klase okvira:

```
type
  TFrame3 = class (TFrame)
    ...
  public
    procedure CreateWnd; override;
```

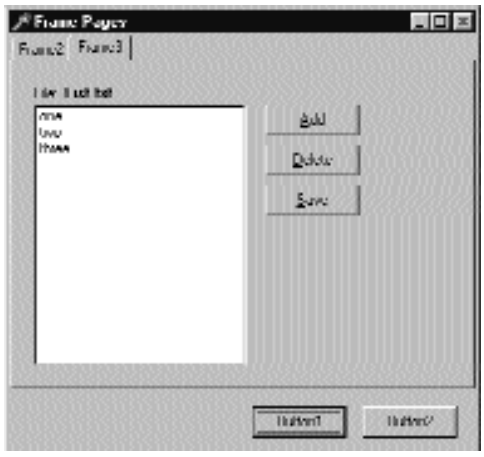
U okviru metoda `CreateWnd` jednostavno možemo iz fajla učitati sadržaj liste.

Više okvira bez strana

Drugi način je da izbegnete kreiranje svih strana uz formular koji treba da ih sadrži. Ovo se može postići tako što će komponenta PageControl ostati prazna i tako što ćete kreirati okvire samo kada se strana prikazuje. Zapravo, kada imate okvire na više strana komponente PageControl, prozori okvira se kreiraju samo kada se prikaže prva strana, što se može uočiti kada se načini pauza u kodu kreiranja poslednjeg primera.

Još radikalniji pristup je mogućnost da se rešite strana i upotrebite TabControl. Kada se koristi na ovaj način, kartica nema odgovarajuće strane, ali tada može da prikaže samo po jednu informaciju. Zbog toga će biti potrebno da kreirate okvir i da uklonite prethodni ili da ga jednostavno sakrijete određivanjem vrednosti `False` za svojstvo `Visible`, ili da pozovete `BringToFront` za novi okvir. Mada ovo izgleda kao mnogo posla, ova tehnika se isplati u velikim aplikacijama jer se smanjuje upotreba resursa i memorije.

Da bih demonstrirao ovakav pristup, izradio sam primer sličan prethodnom, ali koji je ovog puta zasnovan na komponenti TabControl i dinamičkom kreiranju okvira. Glavni formular, koji u vreme izvršavanja možete videti na slici 8.13, sadrži samo TabControl sa po jednom stranom za svaki okvir:



SLIKA 8.13 *Prva strana primera FrameTab u vreme izvršavanja. Okvir unutar kartice je kreiran u vreme izvršavanja.*

```
object Form1: TForm1
  Caption = 'Frame Pages'
  OnCreate = FormCreate
  object Button1: Tbutton . . .
  object Button2: TButton . . .
  object Tab: TTabControl
    Anchors = [akLeft, akTop, akRight, akBottom]
      Tabs.Strings = (
        'Frame2'
        'Frame3')
      OnChange = TabChnage
    end
end
```

Ja sam za svaku karticu načinio zaglavlje koje odgovara nazivu okvira, jer ću ovu informaciju koristiti za kreiranje novih okvira. Kada je formular kreiran, i svaki put kada korisnik promeni aktivnu karticu, program uzima aktuelno zaglavlje kartice i prosleđuje ga metodu `ShowFrame`. Kod ovog metoda, prikazanog niže, proverava da li zahtevani okvir već postoji (nazivi okvira u ovom primeru odgovaraju Delphi standardu koji dodaje broj nazivu klase), a zatim ga premešta u prvi plan. Ukoliko okvir ne postoji, program koristi naziv okvira da bi pronašao odgovarajuću klasu okvira, kreira objekat te klase i dodeljuje mu nekoliko svojstava. Kod prilično koristi reference klase i tehnike dinamičkog kreiranja (razmatrano u Poglavlju 3):

```
type
  TFrameClass = class of TFrame;

procedure TForm1.ShowFrame(FrameName: string);
var
  Frame: TFrame;
  FrameClass: TFrameClass;
begin
  Frame := FindComponent (FrameName + '1') as TFrame;
  if not Assigned (Frame) then
  begin
    FrameClass := TFrameClass (FindClass ('T' + FrameName));
    Frame := FrameClass.Create (Self);
    Frame.Parent := Tab;
    Frame.Visible := True;
    Frame.Name := FrameName + '1';
  end;
  Frame.BringToFront;
end;
```

Da bi ovaj kod funkcionisao, ne smete zaboraviti da dodate poziv `RegisterClass` u sekciji inicijalizacije za svaku jedinicu koja definiše okvir.

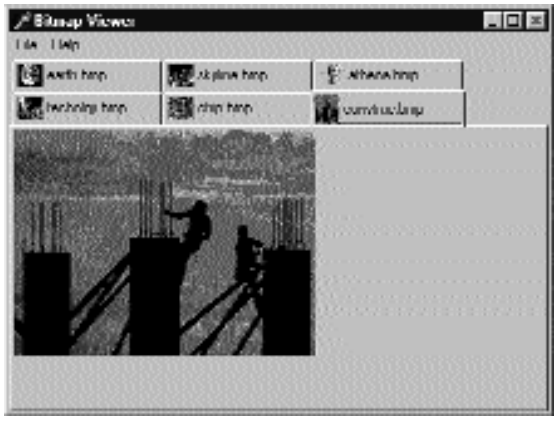
Program za pregled slika u kome vlasnik iscrtava kartice

Upotreba komponente TabControl i dinamičkog pristupa, kakv je prikazan u prethodnom primeru, takođe može da se primeni u opštim (i jednostavnijim) slučajevima. Svaki put kada Vam je potrebno više strana, a sve one imaju istu vrstu sadržaja, umesto da za svaku stranu replicirate kontrole, Vi možete upotrebiti TabControl i promeniti njen sadržaj kada se selektuje nova kartica.

Toj ono što ću ja učiniti u programu za pregled slika sa više strana koji ću Vam prikazati u narednom primeru nazvanom TabOnly. Slika koja se pojavljuje u okviru TabControl ovog formulara, poravnatog sa klijent oblasti, zavisi od selekcije kartica (kao što se može videti na slici 8.14).

Na početku, TabControl sadrži samo lažnu karticu koja opisuje situaciju (Fajl nije selektovan — No file selected). Kada odabere File➥Open, korisnik može odabrati jedan od fajlova prikazanih u okviru za dijalog `File` Open, a niz stringova sa nazivima fajlova (svojstvo Files komponente `OpenDialog1`) se koristi kao tekst za kartice (svojstvo `Tabs` komponente `TabControl1`):

```
procedure TForm1.OpenClick (Sender: TObject);
begin
  if OpenDialog1.Execute then
  begin
    TabControl1.Tabs := OpenDialog1.Files;
    TabControl1.TabIndex := 0;
    TabControl1Change (TabControl1);
  end;
end;
```



SLIKA 8.14 *Interfejs programa za pregled slika iz primera TabOnly. Obratite pažnju na kartice koje iscrtava vlasnik*

Kada prikažemo nove kartice, potrebno je da ažuriramo sliku tako da odgovara prvoj kartici. Da bismo ovo postigli, program poziva metod koji je povezan sa događajem `OnChange` komponente TabControl, kojim se učitava fajl koji odgovara aktuelnoj kartici:

```
procedure TForm1.TabControl1Change(Sender: TObject);
begin
  Image1.Picture.LoadFromFile (
    TabControl1.Tabs [TabControl1.TabIndex])
end;
```

Jedina specijalna karakteristika primera je da je za svojstvo `OwnerDraw` komponente TabControl određena vrednost `True`. To znači da kontrola neće iscrtavati kartice (koje su prazne u vreme dizajniranja) već će to učiniti aplikacija pozivom metoda `OnDrawTab`. Program u svom kodu prikazuje tekst koji je vertikalno centriran upotrebom API funkcije `DrawText`. Tekst koji se prikazuje nije cela

putanja već samo naziv fajla. Zatim, ukoliko je tekst *None*, program čita bitmapu na koju referiše karticu i iscrtava malu verziju na samoj kartici. Da bi se ovo postiglo, progrm koristi objekat `TabBmp`, koji je tipa `Tbitmap`, a kreira se i uklanja zajedno sa formularom. Program, takođe, koristi konstantu `BmpSide` za pravilno pozicioniranje bitmape i teksta:

```
procedure TForm1.TabControl1DrawTab(Control: TCustomTabControl;
  TabIndex: Integer; const Rect: TRect; Active: Boolean);
var
  TabText: string;
  OutRect: TRect;
begin
  TabText := TabControl1.TabS [TabIndex]
  OutRect := Rect;
  InflateRect (OutRect, -3, -3);
  OutRect.Left := OutRect.Left + BmpSide + 3;
  DrawText (Control.Canvas.Handle,
    PChar (ExtractFileName (TabText)),
    Length (ExtractFileName (TabText)),
    OutRect, dt_Left or dt_SingleLine or dt_VCenter);
  if TabText <> 'None' then
  begin
    TabBmp.LoadFromFi1e (TabText);
    OutRect.Left := OutRect.Left - BmpSide - 3;
    OutRect.Right := OutRect.Left + BmpSide;
    Control.Canvas.StretchDraw (OutRect, TabBmp);
  end;
end;
```

Ovaj primer funkcioniše izuzev onda kada selektujete fajl koji ne sadrži bitmapu. Program će upozoriti korisnika standarnim izuzetkom i nastaviće izvršavanje.

Korisnički interfejs čarobnjaka

Baš kao što možete da upotrebite TabControl bez strana, možete da pratite suprotan pristup i upotrebiti PageControl bez kartica. Ono na šta želim da se usredsredim je programiranje korisničkog interfejsa čarobnjaka. Čarobnjakom usmeravate korisnika kroz niz koraka, jedan po jedan ekran, i za svaki korak želite da ponudite mogućnost prelaska na sledeći korak ili povratka na prethodni korak radi korekcije unosa. Dakle, umesto kartica, koje se mogu selektovati u bilo kom redosledu, čarobnjak tipično obezbeđuje kontrole Back i Next. Ovo neće biti složen primer; njegova svrha je da Vam da neka uputstva. Primer je nazvan WizardUI.

Početna tačka je kreiranje niza strana na komponenti PageControl i određivanje vrednosti `False` za svojstvo `TabVisible` svakog TabSheeta (dok se vrednost `True` ostavlja za svojstvo `Visible`). Nasuprot prethodnim verzijama, u Delphiju 5 možete da sakrijete kartice i u vreme dizajniranja. U tom slučaju, potrebno je da koristite iskačući meni kontrole Page ili combo polje u Object Inspectoru da biste prešli na drugu stranu, umesto da koristite kartice. Ali, zašto biste poželei da sakrijete kartice u vreme dizajniranja? Da biste mogli da postavite kontrole na strane i da zatim postavite dodatne kontrole ispred strana (kao što sam ja to uradio u primeru), a da se njihove relativne pozicije ne menjaju u vreme izvršavanja. Takođe, možda želite da uklonite nepotrebna zaglavlja kartica koja zauzimaju mesto u memoriji i resurse aplikacije.

Na prvoj strani, stavio sam sliku i kontrolu na jednu stranu, a na drugu tekst, polje za potvrdu i dve kontrole. Zapravo, kontrola Next je unutar strane, dok je kontrola Back iznad strane (i ovu kontrolu koriste sve strane). Prvu stranu u vreme dizajniranja možete videti na slici 8.15. Naredne strane su slične i na desnoj strani sadrže oznaku, polje za potvrdu i kontrole, dok se na levoj strani ne nalazi ništa.

SLIKA 8.15 *Prva strana primera WizardUI u vreme dizajniranja*

Kada kliknete kontrolu Next na prvoj strani, program proverava status polja za potvrdu i odlučuje koja strana bi trebalo da se prikaže. Ja sam mogao da napišem ovakav kod:

```
procedure TForm1.btnNext1Click(Sender: TObject);
begin
  BtnBack.Enabled := True;
  if CheckInprise.Checked then
    PageControl1.ActivePage := TabSheet2
  else
    PageControl1.ActivePage TabSheen;
  // move image and bevel
  Bevel1.Parent := PageControl1.ActivePage;
  Image1.Parent := PageControl1.ActivePage;
end;
```

Posle aktiviranja kontrole Back program menja aktivnu stranu i premešta grafički deo na novu stranu. Pošto ovaj kod treba ponoviti za svaku kontrolu, ja sam ga smestio u metod pošto sam dodao nekoliko posebnih karakteristika. Evo stvarnog koda:

```
procedure TForm1.btnNext1Click(Sender: TObject);
begin
  if CheckInprise. Checked then
    MoveTo (TabSheet2)
  else
    MoveTo (TabSheet3);
end;

procedure TForm1.MoveTo(TabSheet: TTabSheet);
begin
  // add the last page to the list
```

```
  BackPages.Add (PageControl1.ActivePage);
  BtnBack.Enabled := True;
  // change page
  PageControl1.ActivePage := TabSheet;
  // move image and bevel
  Bevel1.Parent := PageControl1.ActivePage;
  Image1.Parent := PageControl1.ActivePage;
end;
```

Pored koda koji sam već objasnio, metod `MoveTo` dodaje poslednju stranu (stranu pre promene strane) listi posećenih strana, koja se ponaša kao stek. Zapravo, objekat `BackPages` klase `TList` se kreira kada se pokrene program, a poslednja strana se uvek dodaje na kraju. Kada korisnik klikne kontrolu Back, koja ne zavisi ni od jedne strane, program izdvaja poslednju stranu iz liste, uklanja njenu oznaku i prelazi na stranu:

```
procedure TForm1.btnBackClick(Sender: TObject);
var
  LastPage: TTabSheet;
begin
  // get the last page and jump to it
  LastPage := TTabSheet (BackPages [BackPages.Count - 1]);
  PageControl1.ActivePage := LastPage;
  // delete the last page from the list
  BackPages.Delete (BackPages.Count - 1);
  // eventually disable the back button
  BtnBack.Enabled := not (BackPages.Count = 0);
  // move image and bevel
  Bevel1.Parent := PageControl1.ActivePage;
  Image1.Parent := PageControl1.ActivePage;
end;
```

Ovaj kod omogućava korisniku da se vrati unazad nekoliko strana sve dok se lista ne isprazni, a u tom trenutku se deaktivira kontrola Back. Problem kojim mi treba da se pozabavimo je da prilikom pomeranja sa određene strane mi znamo koje su strane "sledeće" a koje "prethodne", ali ono što ne znamo je sa koje strane dolazimo, jer može postojati više puteva kojima se može doći do strane. Sa sigurnošću se možemo vratiti unatrag samo pomoću liste koja prati korake.

Ostatak koda programa, koji prikazuje neke web adrese, je veoma jednostavan. Dobra vest je da možete ponovo da upotrebite strukturu ovog primera u svojim programima i da je potrebno samo da izmenite grafički deo i sadržaj strana.

Dokiranje uz PageControl

Još jedna interesantna karakteristika kontrola Page je specifična podrška dokiranju. Kada dokirate novu kontrolu iznad PageControl, nova strana se automatski dodaje da bi prihvatila kontrolu, što se lako može videti u Delphi okruženju. Da biste ovo postigli, možete jednostavno odrediti PageControl kao domaćina dokiranja i aktivirati dokiranje za klijent kontrole. Ovo najbolje funkcioniše kada imate sekundarne formulare koje želite da prihvatite. Ukoliko želite da celu komponentu PageControl premestite u pokretni prozor i zatim je ponovo dokirate, potreban Vam je panel za dokiranje u glavnom formularu.

To je ono što sam upravo učinio u primeru DockPage koji sadrži sledeći glavni formular:

```
  object Form1: TForm
    Caption = 'Docking Pages'
    object Panel2: TPanel
    Align = alLeft
    AutoSize = True
    DockSite = True
    OnMouseDown = Panel1MouseDown
    object PageControl1: TpageControl
      ActivePage = TabSheet1
      Align = alClient
      DockSite = True
      DragKind = dkDock
      object TabSheetl: TTabSheet
        Caption = 'List'
        object ListBox1: TListBox
          Align = alClient
        end
      end
    end
  end
  object Splitter1: TSplitter
    Cursor = crHSplit
  end
  object Memo1: TMemo
Align = alClient
end
```

Primetićete da je određena vrednost `True` za svojstvo panela `UseDockManager` i da PageControl sigurno sadrži stranu na kojoj se nalazi lista, jer uklanjanje svih strana izvesno izaziva probleme. Program sadrži još dva formulara sa sličnim osobinama (mada sadrže drugačije kontrole):

```
object Form1: TPorm2
  Caption = 'Small Editor'
  DragKind = dkDock
  DragMode = dmAutomatic
  object Memo1: TMemo
    Align = alClient
  end
end
```

Ove formulare možete prevući na PageControl da biste dodali nove strane, koje će imati zaglavlja koja odgovaraju naslovima formulara. Takođe, možete ukloniti dokiranje za ove kontrole pa čak i za ceo PageControl. Da bi se to postiglo, program ne omogućava automatsko prevlačenje, koje bi promenu strana učinilo nemogućom. Umesto toga, funkcija se aktivira kada korisnik klikne oblast PageControl koja ne sadrži kartice, to jest, panel koji se nalazi ispod:

```
procedure TForm1.Panel1MouseDown (Sender: TObject; Button; TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  PageControl1.BeginDrag (False, 10);
end;
```

Ovakvo ponašanje možete testirati pokretanjem primera DockPage, mada je to pokušano da Vam se dočara slikom 8.16. Primetićete da kada uklonite PageControl sa glavnog formulara, možete

direktno dokirati ostale formulare na panel i zatim podeliti oblast ostalim kontrolama. To je situacija koja je prikazana na slici.

SLIKA 8.16 *Glavni formular primera DockPage pošto je formular dokiran na stranu. Primtićete da drugi formular koristi deo oblasti panela koji prihvata prvi panel.*

Kreiranje MDI aplikacija

Pored upotrebe okvira za dijalog, ili sekundarnih formulara, i smeštanja komponenata na formular, postoji treći pristup koji se obično koristi za Windows aplikacije: MDI (Multiple Documet Interface — interfejs za više dokumenata). MDI aplikacija se sastoji od velikog broja formulara koji se prikazuju unutar jednog glavnog formulara.

Ukoliko koristite Windows Notepad, možete otvoriti samo jedan dokument, jer Notepad nije MDI aplikacija. Ali, kada koristite svoj omiljeni tekst-procesor, verovatno možete otvoriti veliki broj različitih dokumenata, a svaki će biti u svom prozoru jer je to MDI aplikacija. Svi ti prozori dokumenata se obično nalaze unutar okvira (frame) ili aplikacije (application) prozora.

U Windowsu 3 i 3.1 Microsoft se plašio upotrebe MDI-ja. Pojavom Windowsa 95 Microsoft je morao da prizna da se većina korisnika ne oseća lagodno sa ovakvim interfejsom. Office 2000 je prvi veliki paket aplikacija koji napušta MDI model u korist modela SDI (Single Document Interface — intefejs za jedan dokument), koji koristi Windows Resource Explorer i ceo operativni sistem. MDI nije mrtav i ponekad može biti koristan model, ali izgleda da formulari sa više strana i formulari koji se mogu dokirati trenutno imaju veću popularnost.

MDI u Windowsu: tehnički profil

Ovaj odeljak sadrži kratak pregled modela MDI, u tehničkim terminima Windowsa. Na trenutak zaboravite Delphi, a ja ću pokušati da Vam dam ideju o tome šta je MDI zapravo (ne kako izgledaju MDI aplikacije). Ukoliko nikada niste izradili MDI aplikaciju i želite kratak uvod, možete i preskočiti ovaj odeljak.

MDI struktura automatski daje programeru mnoge prednosti. Na primer, Windows rukuje listom dete-prozora u jednom od menija MDI aplikacije, i postoje specifični Delphi metodi koji aktiviraju odgovarajuću MDI funkcionalnost, da bi se poređali dete-prozori. Sledi tehnička struktura MDI aplikacije u Windowsu:

- Glavni prozor aplikacije se ponaša kao okvir ili kontejner. Ovaj prozor zahteva pravilnu strukturu menija i malo specifičnog kodiranja (bar kada se programira upotrebom API-ja).
- Specijalni prozor, poznat kao MDI klijent (MDI client), prekriva celu klijent oblast okvirnog prozora obezbeđujući neke specijalne mogućnosti; na primer, MDI klijent rukuje listom dete-prozora. Mada ovo u početku može izgledati čudno, MDI klijent je jedna od unapred određenih Windows kontrola, kao što je polje za izmene ili lista. Prozor MDI klijent ne sadrži tipične elemente interfejsa prozora, kao što su zaglavlje ili bordura, ali je vidljiv. Zapravo, možete promeniti standardne sistemske boje MDI radne površine (koja se naziva "pozadina aplikacije" — Application Background) na strani Appearance okvira za dijalog Display Properties u Windowsu.
- Postoji veliki broj dete-prozora, istih ili različitih tipova. Ovi dete-prozori se ne smeštaju direktno u okvirni prozor, ali je svaki definisan kao dete MDI klijent prozora, koji je odmah dete okvirnog prozora. (Možemo reći da su dete-prozori "unučići" okvira.)

Kada programirate koristeći Windows API, potrebno je malo rada za izradu i održavanje ove strukture, i potrebno je dodatno kodiranje za pravilno rukovanje menijem. Kao što ćete videti u narednom odeljku, ove zadatke je mnogo lakše obaviti u Delphiju.

Okvir i dete-prozori u Delphiju

Delphi čini programiranje MDI aplikacija lakim, čak i bez upotrebe šablona MDI Application koji Vam je na raspolaganju u Delphiju (pogledajte stranu Application okvira za dijalog File➥New). Potrebno je samo da izradite bar dva formulara; za jedan odredite vrednost `fsMDIForm` za svojstvo `FormStyle`, a za drugi formular odredite vrednost `fsMDIChild` za isto svojstvo. To je sve, ili gotovo sve. Jednostavno podesite ova svojstva u nekom jednostavnom programu, pokrenite taj program i videćete kako se formulari ugnežđuju u tipičnom MDI stilu.

Uopšte govoreći, dete-formular se ne kreira prilikom pokretanja i potrebno je da obezbedite način za kreiranje jednog ili više dete-prozora. To se može postići dodavanjem menija sa elementom New i pisanjem sledećeg koda:

```
procedure TMainForm.New1Click (Sender: TObject);
var
  ChildForm: TChildForm;
begin
  ChildForm := TChildForm.Create (Application);
  ChildForm.Show;
end;
```

U prethodnom kodu, kao i u primeru programa koji ću ubrzo razmatrati, ja sam dva formulara nazvao `MainForm` i `ChildForm`. Druga važna karakteristika je dodavanje Window menija i njegova upotreba kao svojstva `WindowMenu` formulara. Ovaj meni će automatski prikazivati sve postojeće dete-prozore. Naravno, možete odabrati bilo koji naziv za meni, ali Window je standardni naziv.

Ovim jednostavnim operacijama možete izraditi jednostavnu MDI aplikaciju. Da bi program pravilno funkcionisao, možemo dodati broj naslovu bilo kog dete-prozora prilikom njegovog kreiranja:

```
procedure TMainForm.New1Click (Sender: TObject);
var
  ChilForm: TChildForm;
begin
  WindowMenu := Window1;
  Inc (Counter);
  ChildForm := TChildForm.Create (Self);
  ChildForm.Caption := ChildFrom.Caption + ' ' +
    IntToStr (Counter);
  ChildForm.Show;
end;
```

Takođe, možete otvoriti veliki broj dete-prozora, smanjiti ili povećati svaki od njih, zatvoriti ih i koristiti meni Window za prelazak sa jednog na drugi prozor. Ukoliko kreirate više od devet dete-prozora, dodaje se element menija More Windows; kada odaberete ovaj element menija, prikazaće se okvir za dijalog (koji obezbeđuje Windows i koji nije deo Vašeg programa) sa potpunim spiskom dete-prozora.

Pretpostavimo sada da želimo da zatvorimo neke od ovih dete-prozora da bismo oslobodili klijent oblast našeg programa. Kliknite kontrolu Close nekih dete-prozora i oni će biti smanjeni! Šta se dešava? Ne zaboravite da kada zatvorite prozor, Vi ga, zapravo, sakrivate. Zatvoreni formulari u Delphiju još uvek postoje mada nisu vidljivi. U slučaju dete-prozora, njihovo sakrivanje neće biti dobro, jer će MDI meni Window i lista prozora još uvek prikazivati postojeći dete-prozor, čak i kada je sakriven. Zbog toga Delphi jednostavno smanjuje MDI dete-prozore kada pokušate da ih zatvorite. Da bismo rešili ovaj problem, potrebno je da uklonimo dete-prozore prilikom zatvaranja, određivanjem vrednosti `caFree` reference `Action` parametra događaja `OnClose`.

Izrada kompletnog menija Window

Naš prvi zadatak je da definišemo bolju strukturu menija za naš primer. Tipično, meni Window sadrži najmanje tri elementa, nazvana Cascade, Tile i Arrange Icons. Da bismo upravljali komandama menija, možemo upotrebiti unapred definisane metode koji su nam na raspolaganju za vrednost `fsMDIForm` svojstva `FormStyle`:

- Metod `Cascade` kaskadno uređuje otvorene MDI dete-prozore. Dete-formulari su uređeni počevši od gornjeg levog ugla klijent oblasti okvirnih prozora i pomeraju

se ka donjem levom uglu. Prozori preklapaju jedan drugog. Dete-prozori predstavljeni ikonama se takođe uređuju (videti `ArrangeIcons`).

- Metod `Tile` prikazuje MDI dete-prozore jedan pored drugog. Dete-formulari se uređuju tako da se ne preklapaju. Klijent oblast okvirnih prozora se deli na jednake delove za različite prozore, tako da se svi mogu prikazati na ekranu bez obzira na to koliko ih je. Metod `Tile` će takođe urediti dete-prozore predstavljene ikonama. Unapred je određeno horizontalno deljenje, mada možete urediti prozore u više kolona ukoliko imate nekoliko dete-prozora. Unapred određeno deljenje se može promeniti svojstvom `TileMode`.
- Svojstvo `TileMode` određuje kako će funkcionisati procedura `Tile`. Jedine dve mogućnosti su `tbHorizontal`, za horizontalno deljenje, i `tbVertical`, za vertikalno deljenje. Neke aplikacije koriste dve različite komande menija za modove deljenja; ostale aplikacije daju samo komandu Tile menija, ali proveravaju da li je pritisnut taster Shift kada je korisnik odabere. To, u stvari, zbunjuje mnoge korisnike te ćete verovatno želeti da Vaša aplikacija bude jednostavna sa samo jednom opcijom za deljenje.
- Procedura `ArrangeIcons` uređuje sve dete-prozore predstavljene ikonama, počevši od donjeg levog ugla klijent oblasti okvirnog prozora ka gornjem desnom uglu. Otvoreni formulari se ne pomeraju.

Ove procedure i svojstva su korisni za rukovanje menijem Window MDI aplikacije. Na primer, možete napisati sledeći kod:

```
procedure TMainForm.Cascade1Click (Sender: TObject);
begin
  Cascade;
end;
```

Bolje rešenje je da postavite ActionList na formular i dodate joj niz unapred određenih MDI akcija. Odgovarajuće klase su `TWindowArrange`, `TWindowCascade`, `TWindowClose`, `TWindowTileHorizontal`, `TWindowTileVertical` i `TWindowMinimizeAll`. Povezani elementi menija će izvršiti odgovarajuću akciju, a biće nedostupni ukoliko nema dete-prozora. Primer MdiDemo, koji ćemo videti, prikazuje upotrebu MDI akcija između drugih stvari koje pokazuje.

Postoje, takođe, i drugi interesantni metodi i svojstva koja su striktno u vezi sa MDI-jem u Delphiju:

- `ActiveMDIChild` je svojstvo samo za čitanje, dostupno je samo u vreme izvršavanja MDI okvirnog formulara i sadrži aktivan dete-prozor. Korisnik može da promeni ovu vrednost selektovanjem novog dete-prozora, ili program može promeniti vrednost upotrebom procedura `Next` i `Previous`.
- Procedura `Next` aktivira dete-prozor koji sledi aktivni prozor po nekom internom redosledu.
- Procedura `Previous` aktivira dete-prozor koji prethodi aktivnom prozoru po nekom internom redosledu.

- Svojstvo `ClientHandle` sadrži Windows hendl MDI klijent prozora koji prekriva klijent oblast glavnog formulara.
- Svojstvo `MDIChildCount` čuva aktuelni broj dete-prozora.
- Svojstvo `MDIChildren` predstavlja niz dece-prozora. Da biste prolazili kroz sve dete-prozore, možete koristiti ovo svojstvo i svojstvo `MDIChildCount`, recimo u petlji `for`. Ovo može biti korisno za pronalaženje određenog dete-prozora ili za rad na svakom od njih.

Primetićete da je interni redosled dete-prozora obrnut redosledu njihovog aktiviranja. To znači da poslednji selektovani prozor predstavlja aktivni prozor (prvi prozor u internoj listi), prethodno selektovani prozor je drugi u listi, a prvi prozor koji je selektovan je poslednji u listi. Ovaj redosled određuje kako će prozori biti uređeni na ekranu. Prvi prozor u listi je prozor koji se nalazi iznad svih ostalih prozora, dok je poslednji prozor u listi prozor koji se nalazi ispod svih drugih prozora i verovatno je sakriven. Zamislite osu (z-osu) koja iz ekrana polazi ka Vama. Aktivni prozor ima najveću vrednost za z-koordinatu i prekriva ostale prozore. Zbog toga je Windows šema poznata kao z-order (z-redosled).

Primer MdiDemo

Ja sam izradio prvi primer da bih prikazao većinu karakteristika jednostavne MDI aplikacije. MdiDemo je zapravo potpuni MDI editor teksta jer svaki dete-prozor sadrži komponentu Memo i može da otvara i čuva tekst-fajlove. Dete-formular sadrži svojstvo `Modified` koje se koristi da bi se naznačilo da li je tekst promenjen. Koristi se prilikom operacija sa fajlovima i prilikom zatvaranja formulara. Operacije sa fajlovima se obavljaju pomoću nekoliko dodatnih metoda, kao što možete videti iz deklaracije klase:

```
type
  TChildForm = class(TForm)
    Memo1: TMemo;
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure Memo1Change(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);
  private
    fModified: Boolean;
    procedure SetModified(const Value: Boolean);
  public
    procedure Load (FileName: string);
    procedure Save;
    property Modified: Boolean
      read FModified write SetModified;
  end;
```

SAVET

Kao što smo razmatrali u Poglavlju 3, ukoliko želite da poštujete OOP pravila i obezbedite dobru enkapsulaciju, uvek formularu dodajte svojstva za izvoženje polja umesto da polje bude javno. Karakteristika Delphija 4 Class Completion omogućava lako dodavanje svojstva formularu tako da ne postoji više nijedan razlog zbog kojeg to ne bi trebalo činiti. ■

Zastavica `fModified` je `True` u obradi događaja `OnChange`, a `False` u obradi događaja `OnCreate` formulara. Takođe se postavlja na `False` svaki put kada se učita ili sačuva novi fajl, kao što možete videti u kodu dva metoda fajla:

```
procedure TChildForm.Load (FileName: string);
begin
  Memo1.Lines.LoadFromFile (FileName);
  Caption : = FileName;
  fModified := False;
end;

procedure TChi1dForm.Save;
begin
  Memo1.Lines.SaveToFile (Caption);
  fModified := False;
end;
```

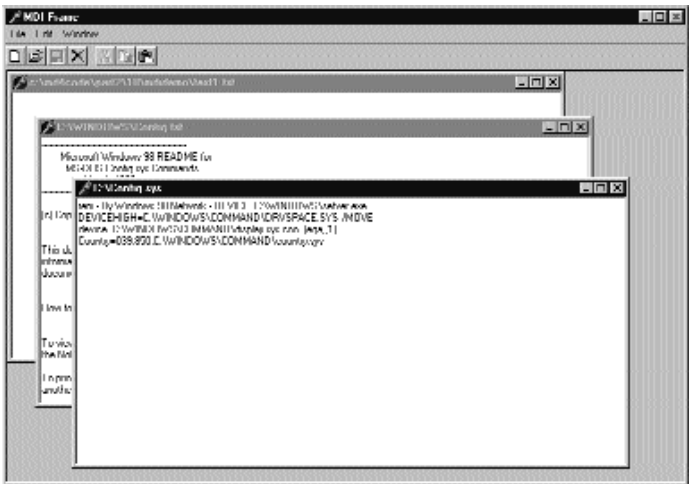
Primetićete da dete-formular koristi zaglavlje za čuvanje naziva fajla — prečica koju sam usvojio umesto da dodam novo svojstvo.

Kada se zatvara dete-formular, proverava se zastavica `fModified`, što možete videti u narednom listingu. Imajte na umu da se metod `OnCloseQuery` dete-formulara takođe automatski aktivira kada zatvarate glavni formular:

```
procedure TChildForm.FormCloseQuery(Sender TObject;
  var CanClose: Boolean);
begin
  CanClose := not fModified or (MessageDlg ('Close without saving',
  mtConfirmation, [mbYes, mbNo), 0) = mrYes);
end;
```

Kao što sam već pomenuo, glavni formular ovog primera je zasnovan na komponenti ActionList. Akcije su dostupne preko nekih elemenata menija i preko palete alata, kao što možete videti na slici 8.17. Detalje komponente ActionList možete videti u izvornom kodu primera.

Sada želim da se posvetim kodu korisničkih akcija. Ponoviću, ovaj kod prikazuje da upotreba akcija čini veoma jednostavnom izmenu korisničkog interfejsa programa bez pisanja dodatnog koda. Zapravo, ne postoji kod koji je direktno vezan za korisnički intefejs.



SLIKA 8.17 *Program MdiDemo koristi niz unapred određenih Delphi akcija povezanih sa menijem i paletom alata*

Jedna od najjednostavnijih akcija je objekat `ActionFont` koji sadrži i obradu `OnExecute`, koja koristi komponentu FontDialog, i obradu OnUpdate, koja onemogućava akciju (i time odgovarajući element menija i kontrolu palete alata) kada nema dete-formulara:

```
procedure TMainForm.ActionFontExecute(Sender: TObject);
begin
  if FontDialog1.Execute then
    (ActiveMDIChild as TChildForm).Memo1.Font :=
      FontDialog1.Font;
end;

procedure TMainForm.ActionFontUpdate(Sender: TObject);
begin
  ActionFont.Enabled := MDIChildCount > 0;
end;
```

Akcija nazvana New kreira dete-formular i dodeljuje unapred određeni naziv fajla. Akcija Open poziva metod `ActionNewExecute` pre učitavanja fajla:

```
procedure TMainForm.ActionNewExecute(Sender: TObject);
var
  ChildForm: TChildForm;
begin
  Inc (Counter);
  ChildForm := TChildForm.Create (Self);
  ChildForm.Caption :=
    LowerCase (ExtractFilePath (Application.Exename)) +
    'text' + IntToStr (Counter) + ' .txt';
  Chi1dForm.Show;
end;

procedure TMainForm.ActionOpenExecute(Sender: TObject);
```

```
begin
  if OpenDialog1.Execute then
  begin
    ActionNewExecute (Self);
    (ActiveMDIChild as TChildForm).Load (OpenDialoq1.FileName);
  end;
end;
```

Metod `Load` formulara zapravo obavlja učitavanje fajla. Slično, metod `Save` dete-formulara koristi akcije Save i Save As. Primetićete da obrada događaja `OnUpdate` akcije Save čini akciju dostupnom samo ukoliko je korisnik izmenio tekst:

```
procedure TMainForm.ActionSaveAsExecute(Sender: TObject);
begin
  // suggest the current file name
  SaveDialog1.FileName := ActiveMDIChild.Caption;
  if SaveDialog1.Execute then
  begin
    // modify the file name and save
    ActiveMDIChild.Caption := SaveDialog1.FileName;
    (ActiveMDIChi1d as TChi1dForm).Save;
  end;
end;

procedure TMainForm.ActionSaveUpdate(Sender: TObject);
begin
  ActionSave.Enabled := (MDIChildCount > 0) and
    (ActiveMDIChild as TChildForm).Modified;
end;

procedure TMainForm.ActionSaveExecute(Sender: TObject);
begin
  (ActiveMDIChild as TChi1dForm).Save;
end;
```

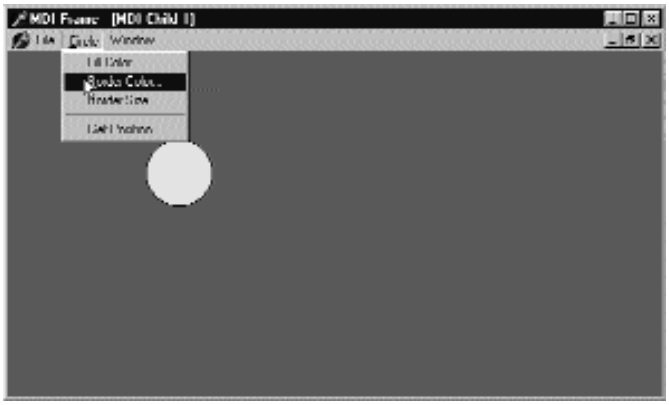
MDI aplikacije sa različitim dete-prozorima

Uobičajeni pristup kod složenih MDI aplikacija je uključivanje dete-prozora različitih vrsta (to jest, zasnovanih na različitim dete-formularima). Mi ćemo izraditi novi primer, koji ćemo nazvati MdiMulti, da bismo istakli neke probleme na koje možete naići kada koristite ovakav pristup. Za ovaj primer biće potrebno da izradimo dva različita tipa dete-formulara. Prvi tip će prihvatiti krug nacrtan na poziciji na koju je poslednji put kliknuto mišem, dok će drugi tip sadržati kvadrat koji skače. Ja ću dodati još jednu karakteristiku glavnom formularu kojom se određuje pozadina, tako što se iscrtavaju slike koje se uklapaju jedna u drugu.

Dete-formulari i meniji

Prvi tip dete-formulara može da prikaže krug na mestu gde je korisnik poslednji put kliknuo jednim od tastera miša. Na slici 8.18 je prikazan primer izlaza programa MdiMulti. Program sadrži meni Circle koji korisniku omogućava da promeni boju površine kruga kao i boju i debljinu njegove bordure. Ono što je interesantno kod programiranja dete-formulara je da ne

moramo da uzmemo u obzir postojanje drugih formulara ili okvirnog prozora. Jednostavno pišemo samo kod formulara i to je sve. Jedino je potrebno da posvetimo pažnju menijima dva formulara.



SLIKA 8.18 *Izlaz primera MdiMulti sa dete-prozorom koji prikazuje krugove*

Ukoliko pripremimo glavni meni za dete-formular, taj meni će zameniti glavni meni okvirnog prozora kada se aktivira dete-formular. MDI dete-prozor, zapravo, ne može da sadrži sopstveni meni. Međutim, činjenica da dete-prozor ne može da ima menije ne treba da Vam smeta jer je to standardno ponašanje za MDI aplikacije. Možete upotrebiti liniju menija okvirnog prozora za prikazivanje menija dete-prozora. Još bolje, mi ćemo spojiti liniju menija okvirnog prozora i odgovarajućeg menija dete-formulara. Na primer, u ovom programu, meni dete-formulara se može smestiti između menija File i Window okvirnog prozora. Ovo možete postići ukoliko koristite sledeće vrednosti za `GroupIndex`:

- Meni File, glavni formular: 1
- Meni Window, glavni formular: 3
- Meni Circle, dete-formular: 2.

Upotrebom ovih vrednosti za indekse menija, linija menija okvirnog prozora će imati ili dva ili tri menija. Prilikom pokretanja linija menija sadrži dva menija. Čim kreirate dete-prozor, imaćete tri menija, a kada se poslednji dete-prozor zatvori (ukloni), nestaće meni Circle. Ovo možete videti na slici 8.18, ali bi trebalo da neko vreme testirate ovo ponašanje izvršavajući program.

Kod dete-prozora jednostavno crta oblik. (Potpuno razmatranje ovog programa možete naći u bonus poglavlju, "Grafika u Delphiju", na adresi `www.sybex.com`.) Ukoliko pogledate izvorni kod, interesantno je primetiti kako se komande menija programa koji se izvršava odnose prema formularima, i da u izvornom kodu svaki formular obrađuje svoje komande, bez obzira na postojanje drugih elemenata.

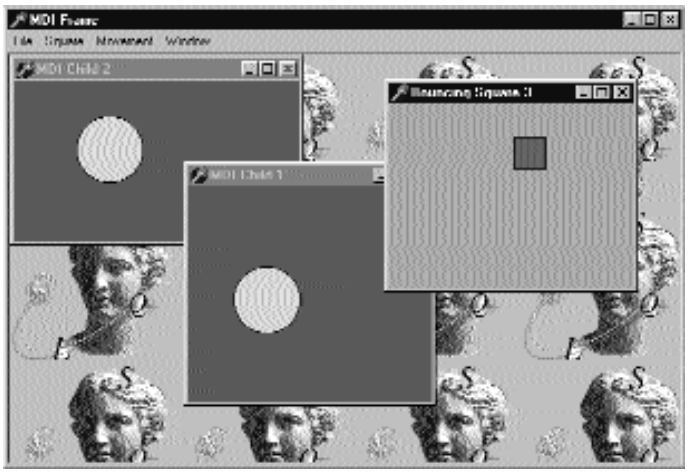
Podaci dete-formulara, naročito koordinate centra kruga, moraju biti deklarisani upotrebom nekih polja formulara, a ne drugim promenljivama deklarisanim unutar jedinice. Zapravo, potrebna nam je određena memorijska lokacija za čuvanje centra kruga za svaki od dete-prozora.

Drugi tip dete-formulara prikazuje pokretnu sliku. Kvadrat, komponenta Shape, se pomera po klijent oblasti formulara u određenim intervalima upotrebom komponente Timer i odbija se od ivica formulara menjajući smer kretanja. Ovaj proces skretanja se određuje prilično složenim algoritmom, za koji nemamo prostora za razmatranje. Suština ovog primera je da Vam pokažem kako se spajanje menija odvija kada imate MDI okvir uz dete-formulare različitog tipa. (Možete proučiti izvorni kod da biste videli kako funkcioniše.)

Menjanje glavnog formulara

Sada je potrebno da integrišemo dva dete-formulara u MDI aplikaciju. Glavni formular mora da obezbedi komandu menija za kreiranje dete-formulara odabranog tipa i za proveru indeksa menija. Meni File sadrži dva odvojena elementa menija New, koji se koriste za kreiranje dete-prozora jednog od tipova. Kod koristi jedan brojač dete-prozora. Kao alternativu možete koristiti dva brojača za dva tipa dete-prozora. Ponoviću, meni Window koristi unapred određene MDI akcije.

Čim se formular ovog tipa prikaže na ekranu, njegov meni se automatski spaja sa glavnom linijom menija. Kada selektujete dete-formular jednog ot tipova, linija menija se menja u skladu sa tim. Kada se jednom zatvore svi dete-prozori, prikazuje se originalni meni. Upotrebom korektnih indeksa mi ćemo Delphiju omogućiti da sve obavi automatski, kao što se vidi na slici 8.19.



SLIKA 8.19 *Linija menija MdiMulti Demo4 aplikacije se automatski menja da bi odgovarala trenutno selektovanom dete-prozoru, kao što možete videti upoređivanjem linije menija sa ove slike i linije menija sa slike 8.18*

Ja sam dodao još nekoliko elemenata menija glavnom formularu. Jedan od elemenata služi za zatvaranje svih dete-prozora, a drugi prikazuje statistiku o dete-prozorima. Metod koji je povezan sa komandom `Count` pretražuje niz `MDIChildren` da bi izbrojao koliko dete-prozora postoji od svakog tipa (upotrebom RTTI operatora `is`). Kada se izračunaju vrednosti, prikazuju se na ekranu funkcijom `MessageDlg`:

```
procedure TMainForm.Count1Click (Sender: TObject);
var
  NBounce, NCircle, I: Integer;
begin
  NBounce := 0;
  NCircle := 0;
  for I := 0 to MDIChildCount - 1 do
    if MDIChildren is TBounceChildForm then
      Inc (NBounce)
    else
      Inc (NCircle);
  MessageDlg (
    Format ('There are %d child forms. '#13 +
      '%d are Circle child windows and' +
      '%d are Bouncing child windows',
      [MDIChi1dCount, NCircle, NBounce]),
    mtINformation, [mbOk], 0);
end;
```

Familija prozora MdiClient

Konačno, program sadrži podršku za prikazivanje slika koje se uklapaju na pozadini. Bitmapa se dobija iz komponente Image i treba je iscrtati na formularu u obradi poruke `wm_EraseBkgnd`. Problem je u tome što ne možemo samo da povežemo kod sa glavnim formularom, jer je njegova površina prekrivena drugim prozorom, prozorom MdiClient koji je opisan ranije u ovom poglavlju.

Ne postoji odgovarajući Delphi formular za ovaj prozor, dakle, kako ćemo obraditi njegove poruke? Moramo se okrenuti Windows tehnici programiranja niskog nivoa poznatoj kao *subclassing*. (Uprkos nazivu, ovo ima malo veze sa OOP nasleđivanjem.) Osnovna ideja je da možemo da zamenimo proceduru prozora, koja prihvata sve poruke prozora, novom procedurom koju mi obezbeđujemo. Ovo se može obaviti pozivanjem API funkcije `SetWindowLong` i obezbeđivanjem memorijske adrese procedure, pokazivača funkcije.

NAPOMENA

Procedura prozora je funkcija kojoj se prosleđuju sve poruke prozora. Svaki prozor mora da ima proceduru prozora i može da postoji samo jedna takva procedura. Čak i Delphi formulari imaju proceduru prozora; mada je sakriveno sistemom, ova procedura poziva virtuelnu funkciju `WndProc` koju možete koristiti. Međutim, VCL ima unapred određenu obradu poruka, koja se zatim prosleđuje metodima za obradu poruka formulara posle nekih pripremnih obrada. Sa svom ovom podrškom, potrebno je da se eksplicitno pozabavite procedurama prozora samo kada radite sa prozorima koji nisu Delphijevi, kao što je ovde slučaj. Za detaljan opis ove teme možete pogledati Priručnik za Delphi programere (Sybex, 1998), kao i neke druge knjige. ■

Izuzev ukoliko nemamo nekakav razlog za promenu unapred određenog ponašanja sistemskog prozora, mi jednostavno možemo da sačuvamo originalnu proceduru i pozovemo je da bismo dobili unapred određenu obradu. Dva pokazivača funkcija, koja se odnose na dve procedure (staru i novu), čuvaju se u dva lokalna polja formulara:

```
private
  OldWinProc, NewWinProc: Pointer;
  procedure NewWinProcedure (var Msg: TMessage);
```

Formular, takođe, sadrži metod koji ćemo koristiti kao novu proceduru prozora uz kod koji se koristi za iscrtavanje pozadine prozora. Kako je ovo metod, a ne procedura prozora, program mora da pozove metod `MakeObjectInstance` da bi metodu dodao prefiks i da bi sistemu omogućio da ga upotrebi kao da je funkcija. Sav ovaj opis se može smestiti u dva složena iskaza:

```
procedure TMainForm.FormCreate(Sender TObject);
begin
  NewWinProc := MakeObjectInstance (NewWinProcedure);
  OldWinProc := Pointer (SetWindowLong (
    ClientHandle, gwl_WndProc, Cardinal (NewWinProc)));
  OutCanvas := TCanvas.Create;
end;
```

Procedura prozora koju mi instaliramo poziva unapred određenu proceduru. Zatim, ukoliko je poruka `wm_EraseBkgnd`, a slika nije prazna, mi ćemo je iscrtati na ekranu mnogo puta upotrebom metoda `Draw`. Ovi objekti crteža se kreiraju kada se pokrene program (pogledajte prethodni kod) i povezuju se sa hendlom koji se prosleđuje kao `wParam` parametar poruke. Uz ovakav pristup mi ne moramo da kreiramo novi objekat `TCanvas` za svaku operaciju iscrtavanja pozadine, čime ćemo uštedeti nešto vremena prilikom čestih operacija. Evo koda koji daje izlaz koji ste videli na slici 8.19:

```
procedure TMainForm.NewWinProcedure (var Msg: TMessage);
var
  BmpWidth, BmpHeight: Integer;
  I, J: Integer;
begin
  // default processing first
  Msg.Result := CallWindowProc (OldwinProc,
    ClientHandle, Msg.Msg, Msg.wParam, Msg.lParam);

  // handle background repaint
  if Msg.Msg = wm_EraseBkgnd then
  begin
    BapWidth := MainForm.Image1.Width;
    BmpHeight := MainForm.Image1.Height;
    if (BmpWidth <> 0) and (BapHeight <> 0) then
    begin
      OutCanvas.Handle := Msg.wParam;
      for I := 0 to MainFormClientWidth div BmpWidth do
        for J := 0 to MainForm.ClientHeight div BmpHeight do
          OutCanvas.Draw (I * BmpWidth,
            J * BmpHeight, MainForm.Image1.Picture.Graphic);
    end;
  end;
end;
```

Šta je sledeće?

Upoznali smo se sa različitim načinima za izradu aplikacije koja sadrži nekoliko formulara ili formulara sa više strana. Videli smo kako možete da kreirate sekundarne neprioritetne formulare ili prioritetne okvire za dijalog. Pored osnovnih primera proučili smo neke napredne teme kao što su: dinamičko kreiranje velikog broja formulara; kreiranje proširenih okvira za dijalog, koristeći uobičajene okvire za dijalog i Delphi okvire za poruke; kreiranje specijalnih okvira About, sa skrivenim porukama ili koristeći ih za uvodne ekrane; MDI tehnike.

Postoji još mnogo stvari koje možemo proučiti da bismo videli kako da izradimo aplikacije sa više formulara i kako da proširimo njihov korisnički interfejs. Ja sam različitim tehnikama posvetio jednaku pažnju, mada su meni neke draže: manje sekundarnih formulara, više okvira za dijalog, MDI samo za specifične programe, beležnice i dokiranje kadgod je to moguće.

Sada možemo preći na veoma aktuelnu Delphi programsku temu: izrada aplikacija za baze podataka. Ovim ćemo se baviti u naredna četiri poglavlja, koja će predstaviti većinu osnovnih tema Delphi programiranja baza podataka. Moguće je napisati knjigu koja se bavi samo ovakvim programiranjem, te opis neće biti detaljan, ali bi trebalo da budete u mogućnosti da dobijete pregled ovog ključnog elementa Delphi programiranja.

Posle ova tri poglavlja o bazama podataka moći ćemo da se posvetimo Delphiju i temama kao što su konstruisanje Delphi komponenata i kontrola ActiveX.

DEO

III

Programiranje aplikacija za baze podataka

U OVOM DELU:

POGLAVLJE

9

Izrada aplikacija za baze podataka

Delphijeva podrška za aplikacije za programiranje baza podataka jedna je od ključnih karakteristika programskog okruženja. Mnogi programeri provode dosta vremena pišući kod za pristupanje podacima koji treba da bude najrobusniji deo aplikacije za baze podataka. Ovo poglavlje sadrži pregled Delphi podrške programiranju baza podataka. Možete kreirati veoma složenu aplikaciju za bazu podataka počinjući svoj rad od praznog formulara ili nekog od Delphijevih Database Form Wizarda.

Ono što u ovoj knjizi nećete pronaći jeste teorija dizajniranja baza podataka. Ja pretpostavljam da već znate osnove dizajniranja baza podataka i da ste već načinili strukturu baze podataka. Neću se baviti problemima specifičnim za baze podataka; moj cilj je da Vam pomognem da razumete kako Delphi podržava ovakvu vrstu programiranja.

Počećemo objašnjenjem kako pristup podacima funkcioniše u Delphiju, a zatim ćemo se upoznati sa komponentama za baze podataka koje su Vam na raspolganju u Delphiju. Neću objašnjavati najjednostavnije primere i neću dati instrukcije korak po korak, recimo kao kada koristite Database Form Wizard, već ću obratiti pažnju na osnove. Neke od tema ovog poglavlja sadrže detaljne primere `TField` komponenata, kreiranja novih tabela uz pomoć Delphi koda i upotrebe grafike. Naredna poglavlja sadrže informacije o mnogim drugim naprednijim temama programiranja baza podataka.

Pristupanje podacima sa i bez BDE

Na kompjuteru se stalni podaci — uključujući i podatke baze podataka — uvek čuvaju u fajlovima. Dva najuobičajenija pristupa su čuvanje cele baze podataka u nečemu što fajl sistemu izgleda kao jedan fajl, i čuvanje svake tabele, indeksa ili bilo kog drugog elementa u odvojenim fajlovima koji se obično nalaze u istom direktorijumu. Delphi podržava oba pristupa, već prema formatu baze podataka koji koristite:

- Tabele Paradox i dBASE definišu baze podataka kao direktorijume i svaka tabela je odvojeni fajl (ili zapravo više fajlova ukoliko uključite indekse i druge fajlove).
- Acces, InterBase i većina SQL servera koriste jedan fajl koji sadrži celu bazu podataka sa svim tabelama i indeksima.

SAVET

Borland Database Engine (BDE) koristi alijas da bi se referisao na fajl baze podataka ili direktorijum. Nove alijase za baze podataka možete definisati upotrebom Database Explorera ili pomoćnog programa Database Engine Configuration. Takođe je moguće definisati ih ukoliko napišete kod koji poziva metode `AddStandardAlias` i `AddAlias` globalnog objekta `Session`, za kojim sledi poziv `SaveConfigFile` da bi se alijas učinio stalnim. Alternativa je BDE funkcija niskog nivoa `DbiAddAlias`. ■

Delphi aplikacije baza podataka nemaju direktan pristup izvoru podataka na koji se referišu i ne mogu direktno manipulisati fajlovima baze podataka. Umesto toga one koriste postojeći mehanizam baze podataka, kao što je Borland Database Engine (BDE) ili Microsoft ActiveX Data Objects (ADO).

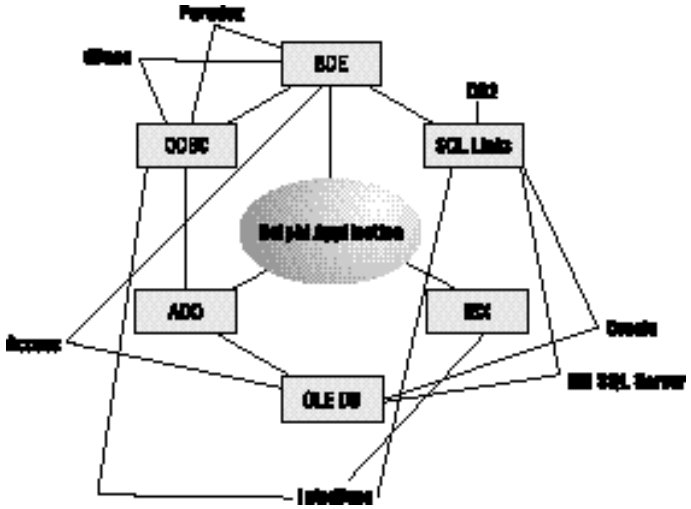
BDE ima direktan pristup velikom broju izvora podataka, uključujući dBASE, Paradox, ASCII, FoxPro, pa čak i Access tabelama. BDE se takođe može koristiti uz Borlandov SQL Links, niz drajvera koji omogućavaju pristup velikom broju lokalnih i udaljenih SQL servera (koje dobijate samo uz Delphi Enterprise). Serveri baza podataka uključuju Oracle, Sybase, Informix, InterBase i DB2. Ukoliko Vam je potreban pristup nekoj drugoj bazi podataka ili nekom drugom formatu podataka, BDE se može povezati sa ODBC drajverima, mada u tom slučaju možete koristiti ADO. Primetićete da BDE obezbeđuje napredne karakteristike (recimo sofisticiranije keširanje i različite varijante spajanja) koje ADO ne nudi.

ADO je Microsoftov interfejs visokog nivoa. ADO je implementiran nad Microsoftovom OLE DB tehnologijom pristupa podacima, koja obezbeđuje pristup relacionim i nerelacionim bazama podataka kao i e-mail i fajl sistemima i uobičajenim radnim objektima. Aplikacije izrađene pomoću Delphi 5 ADO komponenata ne zahtevaju upotrebu BDE biblioteka. Naravno,

korisnici moraju da poseduju izvršni ADO/OLE DB, koji distribuira Microsoft, a deo je operativnog tema Windows 2000. ADO je, takođe, potrebno konfigurisati na mašini korisnika, čak i kada je već instaliran. Poglavlje 12 će potpunije obraditi ADO i odgovarajuće tehnologije.

Delphi Enterprise sadrži osnovne komponente kojima se pristupa Borlandovom InterBase serveru (dostupan je na Delphi instalacionom CD-u; pogledajte Poglavlje 11 za više detalja) i ClientDataSet komponenti (videti Poglavlje 21), koja se može koristiti za pristup lokalnim ili udaljenum podacima. Ove tehnologije daju alternativu tradicionalnoj upotrebi BDE pristupa bazi podataka iz Delphi aplikacija. Slika 9.1 prikazuje alternativne tehnike pristupa podacima koje su Vam na raspolaganju u Delphiju 5, i naznačeno je da su sve komponente za pristup podacima izvedene iz zajedničke osnovne klase `TDataSet`.

Ukoliko se odlučite za tradicionalni BDE pristup (što će biti slučaj sa većinom primera u ovom poglavlju), potrebno je da instalirate BDE uz Vašu aplikaciju na kompjuteru kljenta. To nije teško jer Delphi uključuje "kompaktnu" verziju često korišćenog instalacionog programa (InstallShield) koji se može upotrebiti za pripremu instalacionih diskova za BDE kao i za Vašu aplikaciju. BDE fajlovi su neophodni — Vaše Delphi aplikacije za baze podataka neće funkcionisati bez ovih fajlova — i Vi ih možete slobodno distribuirati.



SLIKA 9.1 *Alternativne tehnologije pristupa podacima koje su Vam na raspolaganju u Delphiju 5*

Delphi komponente za baze podataka

Delphi sadrži veliki broj komponenata koje se odnose na baze podataka. Data Access strana Component Palette sadrži komponente koje se koriste sa bazama podataka BDE orijentisanih aplikacija. Većina ovih komponenata nije vizuelna, jer enkapsuliraju konekcije baze podataka, tabele, upite i slične elemente. Na sreću, Delphi takođe obezbeđuje brojne, unapred određene, komponente koje možete koristiti za prikazivanje i izmenu podataka baze podataka. Na strani

Data Controls postoje vizuelne komponente koje se koriste za prikazivanje i izmene podataka na formularu. Ove kontrole se nazivaju kontrolama koje prepoznaju podatke (data-aware controls).

Da biste pristupili bazi podataka u Delphiju, obično Vam je potreban izvor podataka koji je identifikovan komponentom DataSource. Komponenta DataSource ipak direktno ne identifikuje podatke; ona se referiše na DataSet komponentu. To može biti tabela, rezultat upita, rezultat uskladištene procedure, podaci dobijeni sa udaljenog servera (upotrebom komponente ClientDataSet), Ado, InterBase ili neki drugi skup podataka.

Čim smestite komponentu skupa podataka na formular, možete upotrebiti svojstvo DataSet komponente DataSource da biste se pozvali na skup. Za ovo svojstvo Object Inspector prikazuje sve postojeće skupove podataka aktuelnog formulara ili ostalih formulara i modula podataka koji su povezani sa aktuelnim formularom (upotrebom komande File➥Use Unit).

Tabele i upiti

Najjednostavniji način da navedete pristup podacima u Delphiju je upotrebom komponente Table. Objekat Table se jednostavno referiše na tabelu baze podataka. Kada koristite komponentu Table, potrebno je da naznačite naziv baze podataka koju želite da koristite svojstvom `DatabaseName`. Možete uneti alijas ili putanju direktorijuma u kojem se nalaze fajlovi tabele. Object Inspector će prikazati sve moguće nazive, što zavisi od alijasa instaliranih u BDE-u.

Možete, takođe, naznačiti odgovarajuću vrednost svojstvom `TableName`. Object Inspector će prikazati spisak svih mogućih tabela trenutne baze podataka (ili direktorijuma), te bi trebalo da uvek koristite svojstvo `DatabaseName`.

Drugi skup podataka koji možete koristiti u Delphiju je komponenta Query. Upiti su obično mnogo složeniji od tabele jer zahtevaju komandu SQL jezika. Ipak, upit možete mnogo lakše prilagoditi koristeći SQL nego što možete prilagoditi tabelu (ukoliko znate bar osnovne elemente SQL-a). Komponenta Query sadrži svojstvo `DatabaseName` kao i komponenta Table, ali ne sadrži svojstvo `TableName`. Tabela se određuje SQL iskazom koji se čuva u svojstvu `SQL`.

NAPOMENA

SQL je standardni jezik za pisanje upita nad bazama podataka i, uopšte, za komunikaciju sa bazama podataka. Ukoliko ne baratate dobro SQL-om, opis osnovnih komandi možete naći u Poglavlju 11. Delphi Enterprise sadrži alat za kreiranje SQL upita koji je nazvan SQL Builder i razmatraćemo ga u Poglavlju 11. ■

Na primer, možete napisati jednostavan SQL iskaz kao što je ovaj:

```
select * from Country
```

gde je `Country` naziv tabele, a simbol (*) označava da želite da koristite sva polja iz tabele. Ukoliko dobro baratate SQL-om, možda ćete komponentu Query koristiti mnogo češće, ali efikasnost tabele ili upita zavisi od baze podataka koju koristite. U opštem slučaju, možemo reći da je komponenta Table brža kada se radi sa lokalnim tabelama, dok je komponenta Query brža kada se radi sa SQL serverima, mada ovo nije pravilo i u mnogim slučajevima može imati suprotan efekat. U poglavljima 10 i 11 ću se baviti efikasnošću.

NAPOMENA

Tabela Country koja je pomenuta, odnosi se na fajl COUNTRY.DB koji je deo Delphi primera baze podataka koja se instalira u direktorijumu C:\Program Files\Common Files\Borland Shared\Data. Alias ovog direktorijuma je DBDEMOS koji se određuje prilikom instalacije Delphija. Mnogi od mojih primera u narednim poglavljima će koristiti tabele iz ove Delphi baze podataka. U ostalim primerima ću Vam pokazati kako da kreirate nove tabele, ali ću uglavnom koristiti ovu bazu podataka. ■

Treća komponenta za skupove podataka je StoredProc koja se odnosi na uskladištene procedure SQL server baze podataka. Ove procedure možete izvršiti i rezultate dobiti u obliku tabele. Uskladištene procedure se mogu koristiti samo sa SQL serverima.

Status skupa podataka

Kada vršite operacije nad skupom podataka u Delphiju, možete raditi u različitim stanjima koja su naznačena svojstvom `State` za koje postoji nekoliko različitih vrednosti:

- `dsBrowse` označava da se skup podataka nalazi u normalnom modu pretraživanja koji se koristi za pronalaženje podataka i pretraživanje slogova.
- `dsEdit` označava da se skup podataka nalazi u modu za izmene. Skup podataka prelazi u ovo stanje kada program pozove metod `Edit`, ili kada je određena vrednost `True` za svojstvo `AutoEdit` komponente DataSource, i korisnik počne da menja sadržaj kontrole koja prepoznaje podatke, kao što su kontrole DBGrid ili DBEdit. Kada se promenjeni slog pošalje, skup podataka izlazi iz stanja `dsEdit`.
- `dsInsert` označava da se novi slog dodaje skupu podataka. Ponoviću, ovo se može desiti prilikom poziva metoda Insert, prilikom prelaska na poslednju liniju kontrole DBGrid ili prilikom upotrebe odgovarajuće komande komponente DBNavigator.
- `dsInactive` je stanje zatvorenog skupa podataka.
- `dsSetKey` označava da se pripremamo za pretragu u skupu podataka. To je stanje između poziva metoda `SetKey` i poziva metodima `GotoKey` ili `GotoNearest` (pogledajte primer Search kasnije u ovom poglavlju).
- `dsCalcFields` je stanje skupa podataka za vreme izračunavanja polja, to jest, za vreme poziva obrade događaja `OnCalcFields`. Ponoviću, ovo će biti prikazano u primeru.
- `dsNewValue`, `dsOldValue` i `dsCurValue` su stanja skupa podataka tokom ažuriranja keša.
- `dsFilter` je stanje skupa podataka tokom određivanja filtra; to jest, za vreme poziva obrade `OnFilterRecord` događaja.

U jednostavnim primerima prelazak između ovih stanja se obavlja automatski, ali je važno razumeti ova stanja jer se mnogi događaji odnose na prelazak iz stanja u stanje.

NAPOMENA

Mi ćemo koristiti jednostavan događaj prelaska stanja, događaj `OnStateChange` komponente DataSource u primeru GridDemo, prvog primera ovog poglavlja. ■

Ostale komponente za baze podataka

Pored komponenata Table, Query, StoredProc i DataSource postoje i druge komponente na strani Data Access u Component Palette koje su smeštene na BDE stranu. Ja ću ove komponente opisati u naredna dva poglavlja, ali ovde dajem kratak pregled.

- Komponenta Database se koristi za kontrolu transakcija, zaštitu i kontrolu konekcije. Uobičajeno se koristi za povezivanje sa udaljenim bazama podataka kod klijent/server aplikacija ili da bi se izbeglo preopterećenje veze prema istoj bazi podataka koja se koristi u nekoliko formulara. Komponenta Database se takođe koristi i za određivanje lokalnih alijasa koji se koriste samo u okviru programa. Kada se jednom odredi lokalni alijas za određenu putanju, komponente Table i Query aplikacije se mogu obraćati lokalnom alijasu baze podataka. Ovo je mnogo bolje od repliciranja teškog kodiranja putanje za svaku komponentu DataSet programa.
- Komponenta Session obezbeđuje globalnu kontrolu nad koneckijama baze podataka aplikacije, uključujući i spisak postojećih baza podataka i alijasa kao i događaja za prilagođavanje prijavljivanja na bazu podataka.
- Komponenta BatchMove se koristi za operacije u pozadini, kao što su kopiranje, dodavanje, ažuriranje i uklanjanje vrednosti iz jedne ili više baza podataka.
- Komponenta UpdateSQL Vam omogućava da napišete SQL iskaze koji obavljaju razne akcije ažuriranja nad skupom podataka, kada se koristi upit samo za čitanje (to jest, kada se radi sa složenim upitom). Ova komponenta se koristi kao vrednost za svojstvo UpdateObject za tabele i upite.

Delphi kontrole koje prepoznaju podatke

Videli smo kako se možemo povezati sa izvorom podataka, koristeći bilo tabelu bilo upit, ali još uvek ne znamo kako da prikažemo podatke. Za prikazivanje podataka Delphi obezbeđuje mnoge komponente koje podsećaju na uobičajene Windows kontrole, samo što Delphi kontrole prepoznaju podatke. Na primer, komponenta DBEdit je slična komponenti Edit, a komponenta DBCheckBox odgovara komponenti CheckBox. Sve ove komponente možete pronaći na strani Data Controls u Delphi Component Palette:

- DBGrid ima mogućnost prikazivanja cele tabele odjednom. Omogućava skrolovanje i kretanje, a možete i izmeniti sadržaj. Predstavlja proširenje Delphi Grid kontrola.
- DBNavigator je kolekcija kontrola koje se koriste za kretanje kroz bazu podataka i za izvršavanje akcija nad bazom podataka. Ove kontrole izvršavaju osnovne akcije te ih možete lako zameniti paletom alata.
- DBText prikazuje sadržaj polja koje se ne može menjati. To je grafička kontrola Label koja prepoznaje podatke.
- DBEdit omogućava korisniku da izmeni sadržaj polja (promeni trenutnu vrednost) koristeći kontrolu Edit.

- DBMemo omogućava korisniku da prikaže i izmeni veliko polje za tekst, koje se čuva u Memo ili BLOB polju (Binary Large Object — veliki binarni objekat). Podseća na Memo komponentu.
- DBImage je proširenje komponente Image kojim se prikazuje slika koja se čuva u BLOB polju.
- DBListBox i DBComboBox omogućavaju korisniku da odabere jednu vrednost iz određenog skupa. Ukoliko je taj skup izdvojen iz neke druge tabele baze podataka ili je rezultat upita, trebalo bi da umesto ovih komponenata koristite komponente DBLookupListBox ili DBLookupComboBox.
- DBCheckBox se može koristiti za prikazivanje opcije koja odgovara Boolean polju, ili za njeno uključivanje ili isključivanje, a ova komponenta je proširenje komponente CheckBox.
- DBRadioGroup obezbeđuje niz opcija sa velikim brojem eksluzivnih opcionih kontrola kao što je kontrola RadioGroup.
- DBRichEdit je komponenta koja omogućava korisniku da izmeni formatirani tekstualni fajl; zasnovana je na kontroli RichEdit Windowsa 95.
- DBCtrlGrid je tabela sa više slogova koja može sadržati veliki broj drugih kontrola koje prepoznaju podatke. Ove kontrole se dupliraju za svaki slog skupa podataka.
- DBChart je grafička kontrola i predstavlja verziju komponente Chart.

Sve ove komponente su povezane sa izvorom podataka preko odgovarajućeg svojstva, svojstva `DataSource`. Neke od ovih komponenata se odnose na ceo skup podataka, kao komponente DBGrid ili DBNavigator, dok se druge odnose na određeno polje izvora podataka, što je naznačeno svojstvom `DataField`. Kada odredite svojstvo `DataSource`, svojstvo `DataField` će prikazati spisak vrednosti u combo polju Object Inspectora.

Prilagođavanje tabele za bazu podataka

Naš prvi primer, nazvan GridDemo, koristi tabelu COUNTRY.DB baze podataka DBDEMOS koja sadrži spisak zemalja novog sveta i njihove glavne gradove i broj stanovnika. Jednostavno postavite komponente Table, DataSource i DBGrid na formular i poverite ih. Ukoliko odredite vrednost `True` za svojstvo `Active` tabele, podaci će se u formularu prikazati u vreme dizajniranja. (Ova tehnika se obično naziva živim dizajniranjem podataka — live-data design.) Kada tabela prikazuje ovakve podatke, možete čak koristiti i klizače da biste videli slogove.

U ovom trenutku već možete pokrenuti program, pa čak i izmeniti podatke tabele baze podataka čineći tako izmene trajnim. Ovo je moguće jer komponenta DBGrid sadrži svojstvo `Options` koje uključuje zastavicu `dgEditing` i svojstvo `ReadOnly` za koje je određena vrednost `False`. Ovaj program Vam takođe omogućava da umetnete slog na zadatu poziciju tako što ćete pritisnuti taster Insert, da pridodate novi slog na kraj tabele kada pređete na poslednji slog i pritisnete taster ↓, i da uklonite slog pritiskom kombinacije tastera Ctrl + Del.

SAVET

Koristite ovaj program neko vreme, testirajući kako funkcioniše kada se uključe ili isključe određene zastavice svojstva `Options`. Ove zastavice određuju ponašanje tabele. Opis različitih opcija možete videti u Delphijevom Help fajlu. ■

Pored svojstva `Options`, komponentu DBGrid možete prilagoditi upotrebom svojstva `Columns`. Ovo svojstvo je kolekcija tako da možete odabrati jedan od elemenata liste i zatim podesiti svojstvo u Object Inspectoru, kao što možete videti na slici 9.2.

Lako možete odabrati polja tabele koju želite da prikažete DBGrid komponentom kao kolone, a zatim možete odrediti brojna svojstva kolone (boju, font, širinu, poravnanje i tako dalje) za svako polje, i možete odrediti svojstva zaglavlja, kao što su font i boja. Na ovaj način lako možete prilagoditi tabelu na mnogo načina. Neka od naprednijih svojstava, kao što su `ButtonStyle` i `DropDownRows`, mogu se koristiti za obezbeđivanje editora ćelija tabele ili liste.

SLIKA 9.2 *Možete izmeniti svojstva Columns za DBGrid selektovanjem jedne od kolona u editoru kolekcije i upotrebom Object Inspectora*

U primeru GridDemo ja sam promenio zaglavlje prve kolone i promenio sam font prve i treće kolone. Takođe sam odabrao tamnosivu pozadinu i beli font za prvu kolonu, a uneo sam i nazive nekoliko kontinenata u listu `PickList` polja Continent. Rezultat možte videti na slici 9.3.

SLIKA 9.3 *DBGrid primera GridDemo sadrži nekoliko prilagođenih kolona, uključujući i PickList za njihov sadržaj*

NAPOMENA

Primetićete da kada definišete svojstvo `Columns` komponente DBGrid, možete da promenite veličinu kolona u vreme dizajniranja tako što ćete prevući linije koje ih odvajaju. Isto se može učiniti i u vreme izvršavanja, i može se podesiti, kao i mnoge druge sposobnosti, upotrebom svojstva `Options`. ■

Da bismo rezimirali karakteristike ovog primera, evo dela fajla sa opisom formulara:

```
object Form1: TForm1
  ActiveControl = DBGrid1
  Caption = 'Grid Demo'
  object DBGrid1: TDBGrid
    Align = alClient
    DataSource = DataSource1
    Columns = <
      item
        Alignment = taRightJustify
        Color = clBtnShadow
        FieldName = 'Name'
        Font.Style = [fsBold]
        ReadOnly = True
        Title.Alignment = taRightJustify
        Title.Caption = 'Country'
        Title.Font.Style = [fsBold]
      end
      item
        FieldName = 'Capital'
      end
      item
        Expanded = False
        FieldName = 'Continent'
        Font.Style = [fsItslic]
        PickListStrings = (
          'Africa'
          'Asia'
          'Australia'
          'Europe'
          'North America'
          'South America'')
      end
      item
        FieldName = 'Area'
      end
      item
        FieldName = 'Population'
      end>
    end
    object Table1: TTable
      Active = True
      DatabaseName = 'DBDEMOS'
      TableName = 'COUNTRY.DS'
    end
    object DataSource1: TDatsSource
      DataSet = Table1
      QnStateChange = DataSource1StateChange
    end
  end
```

Status Table

Postoje mnoge stvari kojima možete prilagoditi tabelu, i mi ćemo neke od njih upoznati u narednom poglavlju, u kome ćemo, takođe, razmatrati načine dodavanja grafike. Za sada želim da dodam novu karakteristiku (i nešto koda) ovom primeru. Ukoliko pogledate zaglavlje formulara prikazanog na slici 9.3, primetićete nešto novo: naslov formulara odgovara statusu komponente Table. Kako dobijamo ovakvu informaciju? Jednostavno, obradom `OnStateChange` događaja komponente DataSource. U ovoj obradi događaja primer GridDemo prikazuje trenutni status koji se utvrđuje jednostavim case iskazom:

```
procedure TForm1.DataSource1StateChange(Sender; TObject);
var
  Title: string;
begin
  case Table1.State of
    dsBrowse: Title := Browse;
    dsEdit: Title := 'Edit';
    dslnsert: Title := Insert;
  else
    Title := 'Other state'
  end;
  Caption := 'Grid Oemo - ' + Title;
end;
```

Kod uzima u obzir samo tri stanja komponente Table ovog programa u kojima može da se nađe kada korisnik radi sa odgovarajućim DBGridom.

Kontrole koje prepoznaju podatke, a odnose se na polja

Primer GridDemo dobro funkcioniše, ali mi želimo da isprobamo i druge kontrole, kao što su polja za izmene, i želimo da vidimo određene, a ne sve podatke naše baze podataka. Pre nego što upoznamo suštinu VCL strukture baze podataka, proučvanjem `TField` komponenata, ja želim da pokažem upotrebu nekih kontrola koje se mogu koristiti za prikazivanje i izmene vrednosti polja baze podataka. Polazna tačka je upotreba polja za izmene.

Upotreba DBEdit kontrola

Naredni primer, nazvan EditDemo, koristi nekoliko DBEdit komponenata i oznaka uz tabelu i izvor podataka. Takođe je potrebno da dodamo potpuno novu komponentu DBNavigator. Na slici 9.4 je prikazan formular primera EditDemo u vreme dizajniranja.

Ponoviću, potrebno je da povežemo kontrole koje prepoznaju podatke sa izvorom podataka određivanjem njihovog svojstva `DataSource`, a takođe moramo naznačiti polje za svako od polja za izmene svojstvom `DataField` (Name, Capital i Continet su polja koja se koriste u ovom primeru). Ukoliko ste već povezali izvor podataka sa tabelom i polja za izmene sa izvorom podataka, jednostavno možete odabrati polje iz liste, koja je prikazana u Object Inspectoru, za svojstvo `DataField`. Kada se uspostavi veza, ukoliko je određena vrednost `True` svojstva `Active` komponente Table, vrednosti prvog sloga će se automatski prikazati u poljima za izmene (videti sliku 9.4).

SLIKA 9.4 *Tri komponente DBEdit i komponenta DBNavigator primera EditDemo*

Drugi korak koji možemo izvršiti jeste da onemogućmo neke kontrole komponente DBNavigator uklanjanjem nekih elemenata skupa `VisibleButtons`.

SAVET

Ukoliko uključite svojstvo `ShowHint`, navigator će prikazati različite oblačiće za svaku kontrolu. Možete obezbediti sopstvene oblačiće za svaku od kontrola ukoliko upotrebite `Hints` listu stringova. Stringovi koje unosite se koriste za kontrole po redu: prvi string se koristi za prvu kontrolu, drugi za drugu i tako dalje. Ukoliko neke od kontrola nisu vidljive, možete uneti prazan string. ■

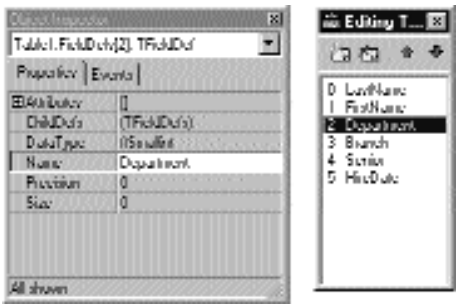
U programu EditDemo ja sam koristio samo neke kontrole onemogućavajući operacije uklanjanja i osvežavanja. Takođe, navigator sam poravnao sa vrhom formulara i odredio sam vrednost `True` za svojstvo `Flat`. Možete pokrenuti program da biste videli da li se pravilno izvršava, i ponovo pogledati zaglavlje: Ja sam za ovaj program kopirao obradu događaja `OnChangeState` komponente DataSource programa GridDemo.

Primetićete da su — kada se program izvršava na početku ili prilikom prelaska na prvi ili poslednji slog — dve kontrole automatski neaktivne. Ipak, ukoliko prelazite slog po slog ka prvom ili poslednjem slogu, kontrole će biti neaktivne samo ukoliko pokušate da se pomerite sa tih slogova. Navigator (ili skup podataka, da budemo precizniji) samo saznaje da nema više slogova u tom smeru. Ostale kontrole su automatski aktivne ili neaktivne kada ulazite ili izlazite iz stanja izmena.

Kreiranje tabele baze podataka

Pre nego što pređemo na ostale kontrole koje prepoznaju podatke, potrebno je da izvršimo još jedan korak. U prva dva primera ovog poglavlja ja sam koristio postojeće tabele baze podataka DBDEMOS, ali za naredne primere potrebno je da kreiram tabelu koja sadrži specifične tipove polja. Zbog toga ću predstaviti temu na koju ćemo se kasnije vratiti: kreiranje novih tabela baze podataka.

Počevši od verzije 4, Delphi Vam omogućava da definišete polja tabele — internu strukturu tabele — u vereme dizajniranja, upotrebom editora kolekcije svojstva `FieldDefs`. Vrednosti koje sam ja upotrebio možete videti na slici 9.5.



SLIKA 9.5 *Editor definicija kolekcije polja*

Kada ste definisali polja, možete kliknuti desnim tasterom miša komponentu tabele i odabrati komandu Create Table. Na ovaj način se kreira nova tabela u vreme dizajniranja. U ovom specifičnom primeru to nije potrebno uraditi jer će program kreirati tabelu prilikom pokretanja, izuzev ukoliko tabela već ne postoji:

```
procedure TForm1.FormCreate (Sender: TObject);
begin
  if not Table1.Exists then
    Table1.CreateTable;
  Table1.Open;
end;
```

Da bi ovaj kod radio, komponenta Table mora da sačuva definicije polja u DFM fajlu pored ostalih svojstava. Ovo se obavlja smo ukoliko odredite vrednost `True` za svojstvo `StoreDefs`. U tabeli 9.1 možete videti definicije polja tabele, a naredni listing prikazuje inicijalni deo odgovarajućih definicija DFM fajla.

Tabela 9.1: polja tabele Workers baze podataka.

Naziv	Tip podataka	Veličina
LastName	ftString	20
FirstName	ftString	20
Department	ftSmallint	
Branch	ftString	20
Senior	ftBoolean	
HireDate	ftDate	

```
object Table1: TTable
  FieldDefs = <
    item
      Name = 'LastName'
      DataType = ftString
      Size = 20
    end
    item
      Name = 'Department'
      DataType = ftSmallint
    end
```

Ova nova tabela baze podataka, nazvana Workers, namenjena je čuvanju nekih podataka o službenicima kompanije. (Primetićete, da smo je nazvali "Employee", to bi dovelo do konflikta naziva ili zamene sa nekom od unapred određenih tabela.)

NAPOMENA

Efekat svojstva `StoreDefs` je mnogo složeniji nego što se to čini na prvi pogled. Ukoliko kliknete formular desnim tasterom miša, primetićete da se u kontekst meniju nalazi opcija Update Table Definition pored očekivanih opcija Delete Table i Rename Table. To znači da definicije polja možete lokalno sačuvati, ali ukoliko se struktura fizičke tabele promeni, tada bi trebalo da ažurirate ovu definiciju. U prethodnim verzijama Delphija definicije polja su se neizostavno učitavale iz tabele baze podataka u vreme izvršavanja; sada ih možete unapred učitati ubrzavajući time otvaranja tabele. Ipak, ukoliko se lokalna i stvarna definicija tabele ne poklapaju, možete upasti u nevolje. ■

Kao što smo videli, primer DbAware kreira tabelu prilikom pokretanja, izuzev ukoliko tabela već nije kreirana. Program zatim otvara tabelu. Da bih izbegao da morate uneti podatke prilikom pokretanja programa, ja sam programu dodao jednostavan metod `AddRandomData`:

```
const
  FirstNames array [1. .10] of string =
    ('John' 'Paul', 'Mark', 'Joseph, 'Bill',
    'Peter', 'Tim', 'Ralph', 'Bob', 'Gary');
  LastNames array [1.. 10] of string =
    ('Ford', 'Qsborse' , 'White', 'MacDonald', 'Lee',
    'Young', 'Parker', Reed', Gates', 'Green');
  NoDept = 3;
  NoBranch = 30;
  NewRecords = 10;

procedure TDbaForm.AddRandomData
var
  I: Integer;
begin
  Randomize;
  for I := 1 to NewRecords do
    Table1.InsertRecOrd ([
      LastNames [Random (Nigh (LastNames)) + 1],
      FirstNames [Random (Nigh (FirstNames)) + 1],
      Random (NoDept) + 1,
      DbComboBox1.Items [Random (NoBranch)]
      Boolean (Random (2)),
      Date - Random (1000)3);
  ShowMessage (IntToStr (NewRecords) ' added')
end;
```

`Metod AddRandomData` poziva metod `InsertRecord` tabele koji nove podatke direktno dodaje — tom prilikom se tabela ne prebacuje u mod za izmene, određivanjem vrednosti polja i slanjem podataka. U ostalim primerima ćemo videti drugačiji pristup za dodavanje podataka tabeli baze podataka. Primetićete da sam za "grananje" polja koristio spisak vrednosti koje postoje u odgovarajućem combo polju.

Prikazivanje alternativnih vrednosti

Sada kada sam kreirao tabelu, mogu je upotrebiti za kreiranje jednostavne demo aplikacije sa kontrolama koje prepoznaju podatke. Na primer, možemo povezati Boolean polje Senior sa kontrolom DBCheckBox. Na ovaj način se korisniku omogućava da promeni status polja tako što će kliknuti kontrolu i prikazati ili ukloniti oznaku.

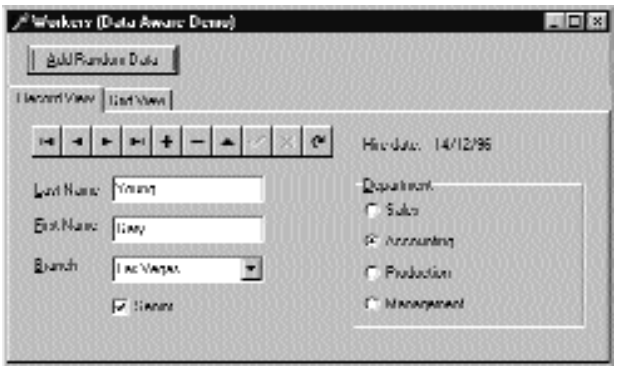
Ovo je prilično trivijalno, dok je upotreba komponenata koje prikazuju alternativne vrednosti nešto komplikovanija. Postoje tri komponente sa ovakvom mogućnostima. To su komponente: DBListBox, DBComboBox i DBRadioGroup. Sve tri komponente omogućavaju selekciju koja korisnika oslobađa unosa i smanjuje mogućnost greške prilikom unosa. Iako su ove tri komponente slične i sadrže spisak stringova u svojstvu `Items`, one se ipak međusobno razlikuju.

- Komponenta DBListBox omogućava selektovanje unapred određenih elemenata ("zatvorena selekcija" — closed selection), ali ne dozvoljava unos teksta i može se koristiti za prikazivanje velikog broja elemenata. Uopšte uzev, najbolje ju je koristiti za prikazivanje šest ili sedam elemenata da bi se izbeglo da zauzima previše prostora na ekranu.
- Komponenta DBComboBox se može koristiti kako za zatvorene selekcije tako i za unos podataka. Takođe, koristi manju površinu formulara jer se njena lista obično prikazuje samo na zahtev.
- Komponenta DBRadioGroup omogućava samo zatvorenu selekciju, treba je koristiti samo za ograničeni broj mogućnosti i omogućava mapiranje values za različite interne vrednosti preko `Values` string liste.

U primeru DbAware ja sam koristio combo polje za selektovanje zemlje i opcione kontrole za selekciju odseka. Ovo je sačuvano u bazi podataka pomoću koda, te sam ih ja ovako mapirao:

```
object DBRadioGroup1: TDBRadioGroup
  Caption = 'Department'
  DataField = 'Department'
  DataSource = DataSource1
  Items.Strings = (
    'Sales'
    'Accounting'
    'Production'
    'Management')
  Values.Strings = (
    '1'
    '2'
    '3'
    '4')
end
```

Primer ovog programa možete videti na slici 9.6. Primetićete da se program zasniva na PageControl: prelaskom na drugu stranu, podatke baze podataka možete videti unutar DBGrida. Ovo bi trebalo da Vam pomogne da shvatite kako se obavlja mapiranje pomoću kontrole RadioGroup. Drugi element je da glavna strana ne omogućava da izmenite datum zapošljavanja koji je prikazan u DBText kontroli samo za čitanje. U narednim primerima ćemo videti kako da obradimo datume.



SLIKA 9.6 *Izlaz programa DbAware koji koristi polje za potvrdu, combo polje i opcione kontrole koje prepoznaju podatke*

Pristupanje poljima sa podacima

Pre nego što pokušamo da izradimo atraktivnije i složenije primere aplikacija, potrebno je da se upoznamo sa još nekoliko tehničkih detalja. Do sada smo koristili sva polja tabela baze podataka. Pretpostavimo da želimo da uklonimo polje ili da dodamo novo polje, recimo polje koje se izračunava. Pri pokušaju da rešimo ove probleme, srećemo se sa opštim pitanjem: kako ćemo pristupiti vrednostima — poljima — trenutnog sloga iz programa? Kako da ih promenimo, a da ih korisnik direktno ne izmeni?

Odgovor na sva ova pitanja leži u konceptu polja (fields). Komponenta Field (instanca klase `TField` ili jedne od njenih potklasa) nije vizuelna komponenta koja je osnovna za svaku Delphi aplikaciju za baze podataka. Kontrole koje prepoznaju podatke su direktno povezane sa ovim Field objektima koji odgovaraju poljima baze podataka.

U dosadašnjim primerima Delphi je automatski kreirao `TField` komponente u vreme izvršavanja. Ovo se dešava svaki put kada program otvori komponentu skupa podataka. Ova polja se čuvaju u nizu `Fields` svojstva tabela i upita, koji predstavlja niz polja. Ovim vrednostima u programu možemo pristupiti preko broja (pristupajući nizu direktno) ili preko naziva (koristeći metod `FieldByName` ili notacije niza):

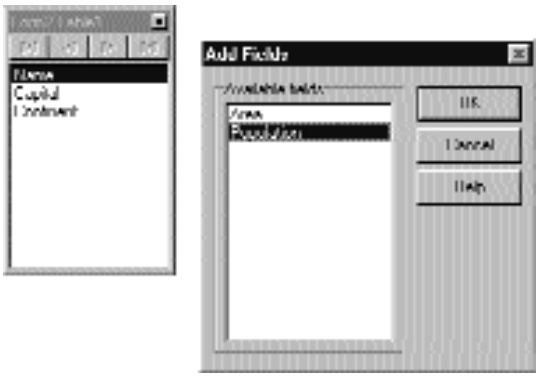
```
Table1.Fields[0].AsString
Table1.FieldByName('Last Name').AsString
Table1.['Last Name'].AsString
```

Alternativno, Field komponente se mogu kreirati u vreme dizajniranja upotrebom Fields editora. U tom slučaju možete odrediti veliki broj svojstava u vreme dizajniranja. Ova svojstva utiču na ponašanje kontrola koje ih koriste, kako za vizuelizaciju tako i za editovanje. Kada nova polja definišete u vreme dizajniranja, ona se prikazuju u Object Inspectoru, baš kao i bilo koja druga komponenta.

NAPOMENA

Mada je Fields editor sličan editorima kolekcija koje koristi Delphi, polja nisu deo kolekcije. To su komponente koje se kreiraju u vreme dizajniranja, prikazane su u published odeljku klase formulara i dostupna su u combo polju na vrhu Object Inspectora. ■

Da biste otvorili Fields editor za tabelu, odaberite Table object, aktivirajte njegov kontekst meni tako što ćete kliknuti desnim tasterom miša i odaberite komandu Fields Editor. Ukoliko dva puta kliknete komponentu Table, dobićete isti efekat. Prikazuje se prazan Fields editor. Sada je potrebno da aktivirate kontekst meni ovog editora. Najjednostavnija operacija koju možete učiniti jeste da odaberete komandu Add koja Vam omogućava da dodate bilo koja polja baze podataka listi polja. Na slici 9.7 je prikazan okvir za dijalog Add Fields, koji prikazuje sva polja koja su Vam na raspolaganju u tabeli. To su polja tabela baze podataka koja se još ne nalaze u listi polja editora.



SLIKA 9.7 *Fields editor sa okvirom za dijalog Add Fields*

Komanda Define Fields editora Vam omogućava da definišete novo polje koje se izračunava, lookup polje ili polje sa modifikovanim tipom. U ovaj okvir za dijalog možete uneti opisni naziv polja koji može sadržati razmake. Delphi generiše interni naziv — naziv komponente polja — koji možete dalje prilagođavati. Zatim odaberite tip podataka za polje. Ukoliko je u pitanju polje koje se izračunava ili lookup polje, a ne samo kopija polja redefinisanog za upotrebu novog tipa podataka, jednostavno potvrdite odgovarajuću opcionu kontrolu. Videćemo kako da definišemo polje koje se izračunava, u odeljku "Dodavanje polja koje se izračunava", i lookup polje u narednom poglavlju.

NAPOMENA

Komponenta `TField` sadrži svojstva `Name` i `FieldName`. Svojstvo `Name` je uobičajeno naziv komponente. Svojstvo `FieldName` je ili naziv kolone tabele baze podataka, ili naziv koji definišete za polje koje se izračunava. Može biti više opisno od svojstva `Name` i dozvoljava upotrebu razmaka. Svojstvo `FieldName` komponente `TField` se kopira u svojstvo `DisplayLabel`, ali naziv ovog polja se može promeniti u bilo koji tekst koji Vam odgovara. Koristi se, između ostalog, za pretraživanje polja metodom `FieldByName` klase `TDataSet` i prilikom upotrebe notacije za nizove. ■

Sva polja koja dodajete ili definišete, uključena su u Fields editor i mogu se koristiti za kontrole koje prepoznaju podatke, ili se mogu prikazati u komponenti Grid. Ukoliko se polje originalne tabele baze podataka ne nalazi u listi, neće moći da mu se pristupi. Kada koristite Fields editor, Delphi dodaje deklaraciju polja kojim se može pristupiti klasi formulara, kao da su u pitanju nove komponente (slično kao što Menu Designer dodaje `TMenuItem` komponente formularu). Komponente `TField` klase ili, preciznije, njenih potklasa su polja formulara, i na ove komponente se možete pozivati direktno iz koda Vašeg programa da biste njihova svojstva promenili u vreme izvršavanja, ili da biste odredili njihove vrednosti, npr. ovakvim izrazom:

```
Table1LastName.AsString
```

U Fields editoru možete, takođe, prevući polja da biste promenili njihov redosled. Dobar redosled polja je naročito važan kada definišete tabelu koja uređuje kolone prema ovom redosledu.

SAVET

Još korisnija krakteristika Fields editora je to što možete prevući polja na površinu formulara i prepustiti Delphiju da automatski kreira odgovarajuće kontrole (kao što su DBEdit, DBMemo ili DBImage). Tip kontrole koja se kreira zavisi od tipa podataka polja i definicija iz Data Dictionary (što ćemo razmatrati u narednom poglavlju). Ovo je veoma brz način za generisanje korisničkih formulara i predlažem Vam da ga isprobate ukoliko to niste ranije činili. Ovo je moj omiljeni način za kreiranje formulara za baze podataka, a mnogo je bolji od upotrebe Database Form Wizarda. ■

Hijerarhija klasa polja

Pre nego što pogledamo primer, utvrdimo znanje upotrebe klase `TField`. Važnost ove komponente se ne sme potceniti. Mada se često koristi u pozaidni, ima osnovnu ulogu u aplikacijama za baze podataka. Kao što sam već pomenuo, čak i kada ne definišete objekte ovog tipa, uvek možete pristupiti poljima (`fields`) tabele ili upita koristeći njihovo svojstvo `Fields`, indeksirano svojstvo `FieldValues` ili metod `FieldByName`. I svojstvo `Fields` i funkcija `FieldByName` kao rezultat daju objekat tipa `Tfield`, tako da ponekad morate da upotrebite operator `as` da biste konvertovali njihove rezultate u potrebni tip (kao `TFloatField` ili `TDateField`) pre nego što pristupite specifičnim svojstvima ovih potklasa.

Primer FieldAcc je prosto proširenje formulara genirasnog pomoću Database Form Wizarda. Ja sam formularu dodao tri točkića na panelu palete alata za pristupanje različitim Field svojstvima u vreme izvršavanja. Prva kontrola menja formatiranje sadržaja kolone tabele. Da bismo to učinili, potrebno je da pristupimo svojstvu `DisplayFormat`, specifičnom svojstvu klase `TFloatField`. Zbog toga je potrebno da napišemo:

```
procedure TForm2.SpeedButton1Click (Sender: TObject);
begin
  (Table1.FieldByName ('Population') as
    TFloatField).DisplayFormat := '###, ###, ###';
end;
```

Kada odredite svojstva polja koja se odnose na unos ili izlaz podataka, izmena će se odnositi na svaki slog tabele. Kada odredite svojstva koja se odnose na vrednost (`value`) polja, Vi se obraćate uvek samo trenutno aktivnom slogu. Na primer, mi možemo prikazati broj stanovnika određene zemlje u okviru za poruke ukoliko napišemo:

```
procedure TForm2.SpeedButton2Click (Sender: TObject);
begin
  ShowMessage (string (Table1 ['Name']) +
    ':' + (Table1 ['Population']));
end;
```

Kada pristupite vrednosti polja, možete upotrebiti niz *As* svojstava da biste odredili vrednost aktuelnog polja koristeći specifični tip podataka (ukoliko je ovo moguće, inače se poziva na izuzetak):

```
AsBoolean: Boolean;
AsDateTime: TDateTime;
AsFloat: Double;
AsInteger: LongInt;
AsString: string;
AsVariant: Variant;
```

Ova svojstva se mogu koristiti za čitanje ili promenu vrednosti polja. Promena vrednosti polja je moguća samo ukoliko se DataSet nalazi u modu za izmene. Kao alternativu za *As* svojstva, vrednostima polja možete pristupiti koristeći svojstvo Value, koje je definisano kao `variant`.

Većina drugih svojstava komponente `TField`, kao što su `Alignment`, `DisplayLabel`, `DisplayWitdth` i `Visible`, odslikavaju elemente korisničkog interfejsa i koriste ih različite kontrole koje prepoznaju podatke, naročito DBGrid. U primeru FieldAcc, kada kliknete treći točkić, menja se `Alignment` svakog polja:

```
procedure TForm2.SpeedButton3Click (Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to Table1.FieldCount – 1 do
    Table1.Fields[I].Aligment := taCenter;
end;
```

Ovo utiče na izlaz DBGrid i DBEdit kontrole koju sam dodao paleti alata, koja prikazuje naziv zemlje. Ovaj efekat možete videti, kao i novi format prikazivanja, na slici 9.8.

SLIKA 9.8 *Izlaz primera FieldAcc pošto su kliknute kontrole Center i Format*

Postoji nekoliko klasa tipova polja u VCL-u. Delphi automatski koristi jedan od njih tako da odgovaraju definiciji podataka baze podataka, kada otvorite tabelu u vreme izvršavanja ili kada koristite Field editor u vreme dizajniranja. Tabela 9.2 sadrži potpuni spisak klasa `TField` klase.

Tabela 9.2: TField potklase (masnim slovima su prikazni novi tipovi polja i odnose se na ADO).

Potklasa	Osnovna klasa	Definicija
TADTField	TObjectField	ADT polje (Abstract Data Type — apstraktni tip podataka), odgovara polju objekta u objektnoj relacionoj bazi podataka.
TAggregateField	TField	Agregatno polje predstavlja sumu. Koristi se u komponenti ClientDataSet i razmatra se u Poglavlju 21.
TArrayField	TObjectField	Niz objekata u objektnoj relacionoj bazi podataka.
TAutoIncField	TIntegerField	Ceo pozitivan broj povezan sa Paradox poljem tabele koje se uvećava (increment field), specijalnim poljem kojem se automatski dodeljuje drugačija vrednost za svaki slog. Primetićete da Paradox AutoInc polja ne funkcionišu uvek kako treba, što ćemo objasniti u narednom poglavlju.
TBCDField	TNumericField	Realni brojevi, sa određenim brojem cifara posle decimalnog zareza.
TBinaryField	TField	Obično se ne koristi direktno. Ovo je osnovna klasa za naredne dve klase.
TBlobField	TField	Binarni podaci bez ograničenja veličine podataka (BLOB je skraćenica za Binary Large Object — veliki binarni objekat). Teoretski maksimum je 2 GB.
TBooleanField	TField	Boolean (logička) vrednost.
TBytesField	TBinaryField	Opcioni podaci velike određene veličine (maksimalno 64K karaktera).
TCurrencyField	TFloatField	Vrednosti valuta, sa istim opsegom kao i novi Real tip podataka.
TDataSetField	TObjectField	Objekat koji odgovara nekoj drugoj tabeli u objektnoj relacionoj bazi podataka.
TDateField	TDateTimeField	Vrednost datuma.
TDateTimeField	TField	Vrednost datuma i vremena.
TFloatField	TNumericField	Brojevi u pokretnom zarezu (8 bajtova).
TGraphicField	TBlobField	Grafika promenljive veličine.
`TGuidField`	TStringField	Polje koje predstavlja COM Globally Unique Identifier, deo ADO podrške.
`TIDispathcField`	TInterfaceField	Polje koje predstavlja pokazivače na IDispatch COM interfejse, deo ADO podrške.
TInetegerField	TNumericField	Celi brojevi u opsegu Long Integer (32 bita).
`TInterfacedFields`	TField	Obično se ne koriste direktno. Ovo je osnovna klasa polja koja sadrže pokazivače na interfejse (IUnknown) kao podatke.

Tabela 9.2: TField potklase (masnim slovima su prikazni novi tipovi polja i odnose se na ADO).

Potklasa	Osnovna klasa	Definicija
TLargeIntField	TIntegerField	Veoma veliki celi brojevi (64 bita).
TMemoField	TBlobField	Tekst promenljive dužine.
TNumericField	TField	Obično se ne koristi direktno. Ovo je osnovna klasa svih numeričkih klasa polja.
TObjectField	TField	Obično se ne koristi direktno. Ovo je osnovna klasa polja koja obezbeđuju podršku za objektne relacione baze podataka.
TReferenceField	TObjectField	Pokazivač na objekat objektne relacione baze podataka.
TSmallIntField	TIntegerField	Celi brojevi u opsegu prirodnih brojeva (16 bitova).
TStringField	TField	Tekstualni podaci određene dužine (8192 bajta).
TTimeField	TDateTimeField	Vrednost za vreme.
TVarBytesField	TBytesField	Opcioni podaci, maksimalno 64K karaktera. Veoma sličan osnovnoj klasi `TBytesField`.
`TVariantField`	TField	Polje koje predstavlja variant tip podataka, deo ADO podrške.
`TWideStringField`	TStringField	Polje koje predstavlja Unicode (16 bitova za svaki karakter) string.
TWordField	TIntegerField	Ceo pozitivan broj u opsegu reči ili neoznačenih celih brojeva (16 bitova).

Mogućnost pristupa bilo kom tipu polja, i odgovarajuće veze sa definicijom podataka, zavisi od baze podataka koju koristite. Na primer, InterBase ne podržava BCD, te nikada nećete dobiti BCDField za tabelu na InterBase serveru. Ovo je naročito tačno za nove tipove podataka koji obezbeđuju podršku objektima relacione baze podataka.

Dodavanje polja koje se izračunava

Sada kada ste se upoznali sa `TField` objektima i videli primer njihove upotrebe u vreme izvršavanja, vreme je da izradimo primer na osnovu deklaracije polja objekata u vreme dizajniranja koristeći Field editor. Možemo početi od prvog primera koji smo izradili, primera GridDemo, i dodati polje koje se izračunava. Tabela `COUNTRY.DB` baze podataka kojoj pristupamo sadrži broj stanovnika i površinu za svaku zemlju, tako da možemo koristiti ove podatke da bismo izračunali gustinu naseljenosti.

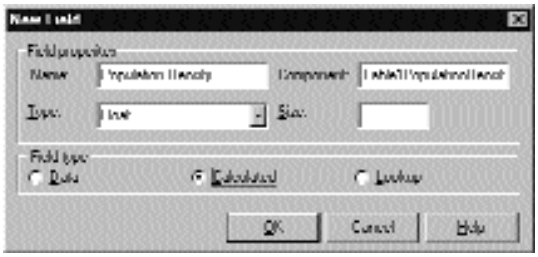
Da biste izradili novi primer, nazvan Calc, selektujte komponentu Table u formularu i otvorite Fields editor (koristeći SpeedMenu formulara). U ovom editoru odaberite komandu Add i selektujte nekoliko polja. (Ja sam odlučio da ih sve upotrebim.) Sada odaberite komandu Define i unesite odgovarajući naziv i tip podataka (`TFloatField`) za novo polje koje se izračunava, kao što možete videti na slici 9.9.

UPOZORENJE

Očigledno je da kada kreirate komponente polja u vreme dizajniranja, koristeći Fields editor, polja koja preskočite neće dobiti odgovarajući objekat. Ono što možda nije očigledno, jeste da polja koja preskočite neće biti dostupna ni u vreme izvršavanja preko svojstava Fields ili `FiledByName`. Kada program otvori tabelu u vreme izvršavanja, ukoliko ne postoje komponente polja u vreme dizajniranja, Delphi kreira objekte polja koji odgovaraju definiciji tabele. Ukoliko postoje polja u vreme dizajniranja, Delphi koristi ta polja i ne dodaje nova. ■

Naravno, takođe je potrebno da obezbedimo način za izračunavanje novog polja. To se postiže događajem `OnCalcFields` komponente Table, koji sadrži sledeći kod:

```
procedure TForm2.TableCalcFields (DataSet: TDataSet);
begin
  Table1PopulationDensity.Value :=
    Table1population.Value / Table1Area.Value;
end;
```



SLIKA 9.9 *Definicija polja koje se izračunava u primeru Calc*

Sve je u redu? Ne baš sve! Ukoliko unesete novi slog i ne odredite vrednosti za broj stanovnika i površinu, ili ukoliko slučajno unesete 0 za površinu, deljenje će dovesti do izuzetka, što će učiniti problematičnim dalje korišćenje programa. Kao alternativu možemo obraditi svaki izuzetak deljenja i dodeliti vrednost 0:

```
try
  Table1PopulationDensity.Value :=
    Table1Population.Value / Table1Area.Value;
except
  on Exception do
    Table1PopulationDensity.Value := 0;
end;
```

Ipak, možemo načiniti i bolje rešenje. Možemo proveriti da li je vrednost za površinu definisana — ukoliko nije null — i da li je nula. Bolje je izbeći upotrebu izuzetaka kada možete predvideti moguće greške:

```
if not Table1Area.IsNull and
    (Table1Area.Value <> 0) then
  Table1PopulationDensity.Value :=
    Table1Population.Value / Table1Area.Value
else
    Table1PopulationDensity.Value := 0;
```

Kod metoda `TableCalcFields` (u bilo kojoj od tri verzije) direktno pristupa nekim poljima. Ovo je moguće jer sam koristio Fields editor, a on je automatski kreirao odgovarajuće deklaracije polja, kao što možete videti u ovom izvodu iz deklaracije interfejsa formulara:

```
type
  TCalcForm = class (TForm)
    Table1: TTable;
    Table1PopulationDensity: TFloatField;
    Table1Area: TFloatField;
    Table1Population: TFloatField;
    Table1Name: TStringField;
    Table1Capital: TStringField;
    Table1Content: TStringField;
    procedure Table1CalcFields (DataSet: TDataset);
    ...
```

Svaki put kada dodate ili uklonite polja u Fields editoru, odmah se može videti efekat Vaših akcija u tabeli formulara. Naravno, nećete videti vrednosti polja koje je izračunato u vreme dizajniranja; te vrednosti možete videti samo u vreme izvršavanja jer su one rezultat izvršavanja Pascal koda.

Kako smo definisali neke komponente za polja, možemo ih upotrebiti da bismo prilagodili vizuelne elemente tabele. Na primer, da bismo odredili format prikazivanja kojim se dodaje zarez za odvajanje hiljada, možemo upotrebiti Object Inspector da bismo izmenili svojstvo `DisplayFormat` i uneli "`###, ###, ###`". Ova izmena se odmah vidi u tabeli u vreme dizajniranja.

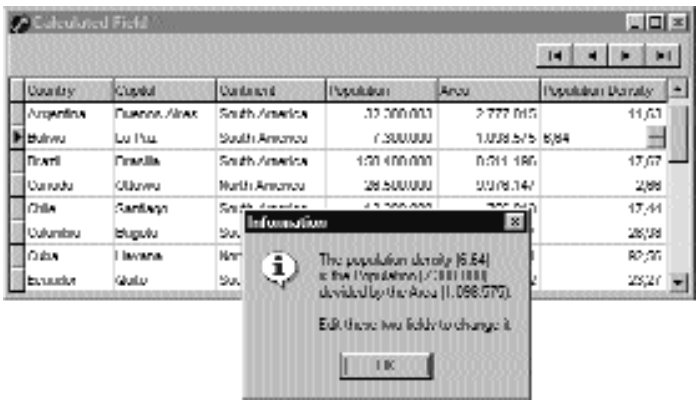
NAPOMENA

Format prikazivanja koji sam upravo pomenuo (i koristio u prethodnom primeru) koristi Windows International Settings za formatiranje izlaza. Kada Delphi prevodi numeričku vrednost ovog polja u tekst, zarez se u stringu formatiranja zamenjuje odgovarajućim karakterom `ThousandSeparator`. Zbog toga će se izlaz programa automatski adaptirati različitim internacionalnim vrednostima. Na kompjuterima koji imaju italijansku konfiguraciju, na primer, zarez će biti zamenjen tačkom. ■

Posle rada sa komponentama tabele i poljima, ja sam prilagodio DBGrid koristeći editor `Columns` svojstva. Odredio sam za kolonu Population Density da bude samo za čitanje i odredio sam vrednost `cbsEllipsis` za svojstvo `ButtonStyle`. Kada odredite ovu vrednost, prikazuje se mala kontrola sa trima tačkama kada korisnik pokuša da izmeni ćeliju tabele. Ukoliko klikne kontrolu, poziva se događaj `OnEditButtonClick` komponente DBGrid:

```
procedure TCalcForm.DBGrid1EditButton (Sender: TObject);
begin
  MessageDlg (Format (
     'The population density (%.2n) '#13 +
     'is the Population (%.0n)'#13 +
     'divided by the Area (%.0n).'#13#13 +
     'Edit these two fields to change it.',
     [Table1PopulationDensity.AsFloat,
     Table1Population.AsFloat,
     Table1Area.AsFloat]),
     MtInformation, [mbOK], 0);
end;
```

Zapravo, ja nisam obezbedio pravi editor, već poruku koja opisuje situaciju, kao što možete videti na slici 9.10, koja prikazuje vrednosti polja koja se izračunavaju. Da biste kreirali editor, možete izraditi sekundarni formular za obradu specijalnih podataka koji se unose.



SLIKA 9.10 *Izlaz primera Calc. Obratite pažnju na polje Population Density koje se izračunava, kontrolu sa trima tačkama i poruku koja se prikazuje kada kliknete kontrolu.*

Pretraživanje i dodavanje polja tabele

TField komponente se mogu koristiti za pristupanje podacima i manipulisanje tabelom u vreme izvršavanja. Mi smo videli samo ograničeni primer direktnog pristupa podacima; u prethodnom primeru koristili smo vrednosti dva polja da bismo izračunali vrednost trećeg polja. Sada ćemo izraditi neke jednostavne primere koji će nam omogućiti da koristimo polja za pretraživanje elemenata tabele, da operišemo sa vrednostima i da pristupamo informacijama koje opisuju tabele baze podataka. Postoji još načina na koje možete upotrebiti komponente polja, ali ovo bi trebalo da Vam da ideju šta sve možete da učinite.

Pronalaženje slogova u tabeli

Za ovaj primer nam je potreban novi formular koji će ovog puta biti povezan sa bazom podataka EMPLOYEE.DB, a koja predstavlja još jedan primer Delphi tabela. Da biste pripremili formular, možete upotrebiti Database Form Wizard ili možete prevući polja iz Fields editora, što je operacija koja će automatski dodati odgovarajuće oznake.

SAVET

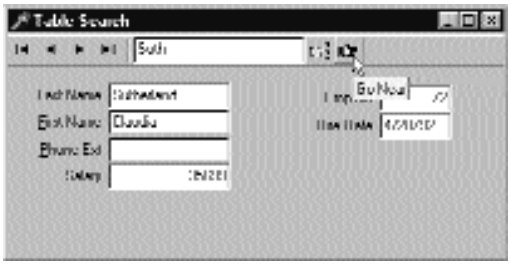
Ukoliko upotrebite polja za izmene koja prepoznaju podatke unutar polja za skrolovanje poravnatog u klijent oblasti, bez ikakvih problema možete menjati veličinu formulara. Kada formular postane previše mali, automatski će se prikazati klizači u oblasti koja sadrži polja za izmene. ■

Umesto unapred određene Delphi navigator komponente, možemo dodati standardnu komponentu Toolbar i povezati kontrole sa unapred određenim akcijama nad skupom podataka koje su nam na raspolaganju u komponenti ActionList. Ja sam formularu dodao ImageList i povezao ga

sa ActionListom da bih omogućio da lista slika dobije slike za standardne akcije. Zatim sam dodao unapred određene ActionList standardne akcije `TDataSetFirst`, `TDataSetLast`, `TDataSetNext` i `TDataSetPrior`, i još dve akcije za kod pretraživanja.

Sada jednostavno možete povezati kontrole palete alata sa odgovarajućim akcijama i paleti alata dodati polje za izmene u koje korisnik može uneti naziv koji traži, kao što se može videti na slici 9.11. Kontrole će izvršiti odgovarajuću akciju kada se na nih klikne, a biće neaktivne kada se nalazite na vrhu ili dnu skupa pdataka.

Mogućnosti pretraživanja se aktiviraju dvema kontrolama koje su povezane sa akcijama. Prva kontrola je povezana sa `ActionGoto` koja se koristi za pronalaženje identične vrednosti, a druga kontrola je povezana sa `ActionGoNear` koja se koristi za pronalaženje najsličnije vrednosti. U oba slučaja želimo da uporedimo tekst polja za izmene sa poljima LastName tabele Employee.



SLIKA 9.11 *Primer pronalaženja najsličnije vrednosti upotrebom aplikacije Search*

Komponenta Table sadrži metode kojima se može obaviti pretraživanje, kao što su metodi `GotoKey`, `FindKey`, `GotoNearest`, `FindNearest` i `Locate`. Metod `Locate` koristi optimalan pristup: ukoliko je dostupan indeks, metod ga koristi radi bržeg pretraživanja; inače, obavlja se prosto sekvencijalno pretraživanje. Da biste upotrebili prvu grupu metoda pretraživanja, potrebno je da podesite svojstvo `IndexFieldNames` komponente Table. (U ovom slučaju možete direktno selektovati string *LastName;FirstName* iz liste.)

Metodi Find

Kada je indeks dobro određen, možemo izvršiti pretraživanje. Najjednostavniji pristup je upotreba metoda `FindNearest` za aproksimativno pretraživanje i metoda `FindKey` za pronalaženje tačne vrednosti:

```
// goto
Table1.FindNearest ([EditName.Text]);

// go near
if not Table1.FindKey ([EditName.Text]) then
  MessageDlg ('Name not Found', mtError, [mbOK], 0);
```

Oba Find metoda koriste kao parametre niz konstanti. Svaki element niza odgovara indeksiranom polju. U našem slučaju, mi ćemo proslediti samo vrednost za prvo polje indeksa, tako da se ostala polja neće uzimati u obzir.

Metodi Goto

Metode FindNearest i FindKey je lako koristiti. Da biste bolje razumeli kako oni funkcionišu, možemo pogledati upotrebu metoda GotoNearest i GotoKey. Poslednja dva metoda su, zapravo, veoma bliska BDE pozivima niskog nivoa. Jednostavnije od dva metoda je pronalaženje najsličnije vrednosti kontrole GotoNearest:

```
// goto near
Table1.SetKey;
Table1 ['LastName'] := EditName.Text;
Table1.GotoNearest;
```

Kao što možete videti iz ovog koda, svako pretraživanje nad tabelom se sprovodi u tri koraka. To su: početno stanje pretraživanja nad tabelom, određivanje vrednosti koja se traži i početak procesa pretraživanja pomeranjem trenutnog sloga na zahtevanu poziciju.

Kod koji se koristi za pozivanje drugih metoda pretraživanja, upotrebom algoritma za pronalaženje tačne vrednosti, veoma je sličan. Jedina razlika je u dva iskaza:

```
// goto
Table1.SetKey1;
Table1 ['LastName'] := EditName.Text;
Table1.KeyFieldCount := 1;
If not Table1.GotoKey then
  MessageDlg ('Name not found', mtError, [mbOK], 0);
```

Kao što sam ranije naglasio, za ovaj kod je potreban odgovarajući indeks nad tabelom. Obratite pažnju na vrednost koja je određena za svojstvo KeyFieldCount, koje označava da želim da koristim samo prvo od dva polja koja sačinjavaju indeks. Druga razlika je u tome da je izvršavanje procedure GotoNearest uvek uspešno, čime se pomera kursor na najsličniju vrednost (najsličnija vrednost uvek postoji, čak i kada nije toliko slična). S druge strane, metod GotoKey se ne izvršava uspešno ukoliko ne postoji identična vrednost, a povratnu informaciju možete proveriti i upozoriti korisnika.

FindKey izvršava istovetne korake kao i verzija GotoKey. FindKey i GotoKey obezbeđuju jednaku funkcionalnost, izuzev što se FindKey lakše koristi, a GotoKey nudi bolju obradu grešaka.

Metod Locate

Ukoliko tabela nema indeks nad poljem koje pretražujete (bar ne za lokalne tabele), ne možete koristiti prethodne dve tehnike. Treća, opštija, tehnika je upotreba metoda Locate. Ovaj pristup je zgodan za svaku priliku, jer ukoliko postoji indeks nad poljem koje pretražujete, Locate ga automatski koristi; inače, izvršava obično (i sporije) pretraživanje.

Upotreba metoda Locate je veoma jednostavna: potrebno je da samo obezbedite prvi string sa poljima koja želite da pretražite i alternativu sa vrednošću ili vrednostima koje tražite. Da biste pretražili više polja, potreban Vam je niz vrednosti. (Takav niz možete kreirati pozivom VarArrayCreate.) Evo primera upotrebe metoda Locate, koji je izdvojen iz programa Search:

```
// goto
if not Table1.Locate ('LastName', EditName.Text, []) then
  MessageDlg ('Name not found', mtError, [mbOK], 0);
```

Suma kolone tabele

U našim dosadašnjim primerima korisnik može videti trenutni sadržaj tabele baze podataka i može ručno izmeniti podatke ili umetnuti nove slogove. Sada ćemo videti kako možemo promeniti podatke tabele pomoću koda programa. Ideja ovog primera je prilično jednostavna. Tabela Employee, koju smo koristili, sadrži polje Salary. Menadžer kompanije može zaista pretražiti tabelu i promeniti platu nekog od zaposlenih. Ali, koliki su ukupni troškovi za plate ove kompanije i šta se dešava ukoliko menadžer odluči da svima poveća (ili umanji) platu za 10 procenata?

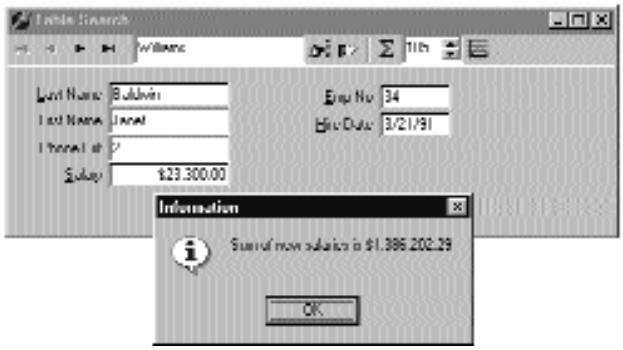
Ovo su dva cilja primera Total, koji predstavlja proširenje prethodnog programa. Paleta alata novog primera sadrži još dve kontrole (i dve odgovarajuće akcije) i komponentu SpinEdit. Postoje i dve manje izmene u odnosu na prethodni primer. Ja sam otvorio Fields Editor tabele i uklonio sam polje `TableSalary`, koje je definisano kao `TFloatField`. Zatim sam odabrao komandu New Field i dodao sam isto polje, sa istim nazivom, ali koristeći tip podataka `TCurrencyField`. Ovo nije polje koje se izračunava; to je jednostavno polje koje je konvertovano u novi (ali ekvivalentni) tip podataka. Upotrebom ovog novog tipa podataka program će dati novi unapred određeni izlaz koji je pogodan za vrednosti valuta.

Sada možemo usmeriti pažnju na kod ovog novog programa. Prvo, pogledajmo kod akcije sumiranja. Ova akcija Vam omogućava da izračunate sumu svih plata, zatim da izmenite neke vrednosti, i da zatim izračunate novu sumu. U osnovi je potrebno da pretražimo tabelu i pročitamo vrednost `Table1Salary` za svaki slog:

```
begin
  Table1.First;
  while not Table1.EOF do
  begin
    Total := Total + Table1Salary.Value;
    Table1.Next;
  end;
end
```

Ovaj kod se izvršava, kao što možete videti na slici 9.12, ali ima mnogo problema. Jedan problem je u tome da se pokazivač sloga pomera na poslednji slog te se gubi prethodna pozicija. Da bismo izbegli ovaj problem, potrebno je da sačuvamo trenutnu poziciju pokazivača sloga tabele i da ga restauriramo kada se akcija izvrši. Ovo se može postići upotrebom *table bookmarka*, specijalne promenljive koja se koristi za čuvanje pozicije sloga u tabeli baze podataka. Tradicionalan pristup je da se deklariše promenljiva `TBookmark` tipa podataka i da se inicijalizuje prilikom čitanja trenutne pozicije iz tabele:

```
var
  Bookmark: TBookmark;
begin
  Bookmark := Table1.GetBookmark;
```



SLIKA 9.12 *Izlaz programa Total kojim se prikazuje suma svih plata zaposlenih*

Na kraju metoda `ActionTotalExecute` možemo restaurirati poziciju i ukloniti obeležje (bookmark) sledećim dvama iskazima:

```
Table1.GotoBookmark (Bookmark);
Table1.FreeBookmark (Bookmark);
```

Još bolje, možemo upotrebiti svojstvo `Bookmark` klase `TDataset`, koje se referiše na obeležje koje se automatski uklanja. (Ovo je implementirano kao opaque string, struktura koja podleže upravljanju postojanja stringa, ali nije string, te ne treba da gledate šta se nalazi unutra.) Evo kako možete izmeniti prethodni kod:

```
var
  Bookmark: TBookmarkStr;
begin
  Bookmark := Table1.Bookmark;
  ...
  Table1.Bookmark := Bookmark;
```

Još jedan sporedni efekat ovog programa je mada ćemo mi ipak restaurirati pokazivač na početnu poziciju — što ćemo možda videti skrolovanje slogova za vreme izvršavanja rutine. Ovo se može izbeći onemogućavanjem kontrola koje su povezane sa tabelom tokom pretraživanja. Tabela sadrži metod `DisableControls` koji možemo pozvati pre nego što počne da se izvršava petlja `while`, i metod `EnableControls` koji možemo pozvati posle izvršavanja petlje, dakle, kada se restaurira pokazivač.

SAVET

Onemogućavanje kontrola koje prepoznaju podatke za vreme izvršavanja dugih operacija ne samo da poboljšava korisnički interfejs (jer se izlaz ne menja konstantno) već i prilično ubrzava izvršavanje programa. Zapravo, vreme koje je potrebno za ažuriranje korisničkog interfejsa mnogo je veće od vremena koje se troši na izračunavanja. Da biste ovo proverili, smestite u komentar metode `DisableControls` i `EnableControls` primera Total i videćete razliku. ■

Konačno, susrećemo se sa nekim opasnim greškama prilikom čitanja podataka tabele, naročito ukoliko program čita podatke sa servera preko mreže. Ukoliko nastanu bilo kakvi problemi prilikom dobijanja podataka, pojavljuje se izuzetak, kontrole ostaju neaktivne, a program ne može nastaviti sa normalnim izvršavanjem. Dakle, trebalo bi da koristimo `try-finally` blok. Zapravo, ukoliko želite

da napravite program koji je 100 procenata siguran od grešaka, trebalo bi da upotrebite dva ugnežđena `try-finally` bloka. Uključujući ovu izmenu i prethodne dve, dobili smo ovakav kod:

```
procedure TSearchForm.ActionTotalExecute (Sender: TObject);
var
  Bookmark: TBookmarkStr;
  Total: Real;
begin
  Bookmark := Table1.Bookmark;
  try
    Table1.DisableControls;
    Total := 0;
    try
      Table1.Fi rst;
      while not Table1.EOF do
      begin
        Total := Total + Table1Salary.Value;
        Table1.Next;
      end;
    finally
      Table1.EnableControls;
  end
  finally
    Table1.Bookmark := Bookmark;
  end;
  MessageDlg ('Sum of new salaries is ' +
    Format (%m , [Total]), mtInformation [mb0K], 0);
end;
```

Ja sam napisao ovaj kod da bih Vam pokazao primer petlje kojom se pretražuje sadržaj tabele, ali imajte na umu da postoji alternativni pristup zasnovan na korišćenju SQL upita kojim se izračunava suma vrednosti polja.

NAPOMENA

Kada koristite SQL server, prednost u brzini pozivanjem SQL-a za izračunavanje sume može biti velika jer nije potrebno da pomerate sve podatke svakog od polja sa servera na klijent. Server klijentu šalje samo konačan rezultat. ■

Menjanje kolone tabele

Kod akcije uvećanja je sličan kodu koji smo upravo videli. Metod `ActionIncreaseExecute` takođe pretražuje tabelu izračunavajuću sumu plata kao i prethodni metod. Mada sadrži samo dva iskaza više, postoji ključna razlika. Kada povećate platu, Vi zapravo menjate podatke tabele. Dva ključna iskaza se nalaze unutar petlje `while`:

```
while not Table1 .EOF do
begin
  Table1.Edit;
  Table1Salary.Value := Round (Table1Salary.Value *
    SpinEdit1.Value) / 100;
  Total := Total + Table1Salary.Value;
  Table1.Next;
end;
```

Prvi iskaz dovodi tabelu u mod izmena tako da izmene nad poljima imaju trenutan efekat. Drugi iskaz izračunava novu visinu plate množeći prethodnu vrednost vrednošću komponente SpinEdit (to je po definiciji 105) i deleći je sa 100. To je povećanje od 5 procenata, mada se vrednosti zaokružuju na najbližu celobrojnu vrednost. Ovim programom možete promeniti visinu plata — možete čak i udvostručiti plate — samo jednim klikom miša.

UPOZORENJE

Primetićete da tabela prelazi u mod edit svaki put kada se izvršava petlja `while`. To je zbog toga što se u skupu podataka operacije izmena mogu obavljati samo nad jednim slogom u jednom trenutku. Morate završiti operaciju izmene pozivanjem `Post` metoda ili pomeranjem na drugi slog kao u prethodnom kodu. Kada želite da promenite drugi slog, morate ponovo preći u mod za izmene. ■

Aplikacije za baze podataka sa standardnim kontrolama

Mada je uopšteno lakše napisati Delphi aplikacije koje koriste kontrole koje prepoznaju podatke, to svakako nije neophodno. Kada je potrebno da zaista imate veoma preciznu kontrolu nad korisničkim interfejsom aplikacije za bazu podataka, možda želite da prilagodite transfer podataka iz objekata polja do vizuelnih kontrola. Moj lični stav je da je ovo neophodno samo u veoma specifičnim slučajevima, jer možete prilagoditi kontrole koje prepoznaju podatke određivanjem njihovih svojstava i obradom događaja objekata polja. Ipak, pokušaj rada bez kontrola koje prepoznaju podatke bi trebalo da Vam pomogne u razumevanju osnovnog ponašanja Delphija, a i pomoći će mi da uvedem neke događaje koji imaju veze sa bazom podataka (razmatraćemo ih u odeljcima "Događaji baze podaka" i "Događaji polja").

Programiranje aplikacije koja ne koristi kontrole koje prepoznaju podatke, može se odvijati na dva različita načina. Možete oponašati standardno Delphi ponašanje, verovatno ostavljajući po strani specifične slučajeve, ili se možete odlučiti za prilagodljiviji pristup. Ja ću prvu tehniku demonstrirati primerom NonAware, a drugu primerom SendToDb.

Oponašanje Delphi kontrola koje prepoznaju podatke

Ukoliko želite da izradite aplikaciju koja ne koristi kontrole koje prepoznaju podatke već se ponaša kao standardna Delphi aplikacija, potrebno je da napišete obrade događaja za operacije koje bi se automatski izvršile da su u pitanju kontrole koje prepoznaju podatke. U osnovi je potrebno da skup podataka prevedete u mod izmena kada korisnik menja sadržaj vizuelnih kontrola, i ažurirate objekte polja skupa podataka kada korisnik napusti kontrole pomerajući fokus na neki drugi element.

SAVET

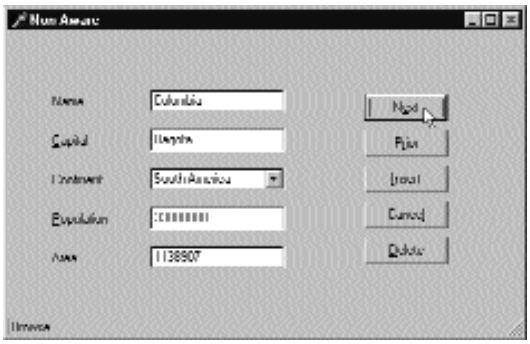
Ovaj pristup je naročito zgodan prilikom integrisanja kontrole koja ne prepoznaje podatke, kao što je DateTimePicker, u standardnu aplikaciju. ■

Drugi element primera NonAware je još jedna lista kontrola koja odgovara nekim kontrolama DBNavigatora. Pet kontrola je povezano sa pet metoda komponente Table: `Next`, `Previous`, `Insert`, `Cancel` i `Delete`. Sledi rezime Delphi fajla formulara:

```
object Form1: TForm
  Caption = 'Non Aware'
  // 5 labels omitted
  object EditName: TEdit
    Text = 'EditName'
    OnExit = EditNameExit
    OnKeyPress = EditKeyPresS
  end
  object EditCapital: TEdit...
  object EditPopulation: TEdit..
  object EditArea: TEdit...
  object ComboContinent: TCamboBox
    Items.Strings = (
      'South America'
      'North America'
      'Europe'
      'Asia'
      'Africa'')
    Text = 'ComboContinent'
    OnDropDown = ComboContinentDropDown
    OnExit = ComboContinentExit
    OnKeyPress = EditKeyPress
  end
  // 5 buttons omitted
  object StatusBar1: TStatusBar
    SimplePanel = True
  end
  object DataSource1: TDataSource
    DataSet = Table1
    OnStateChange = DataSource1StateChange
    OnDataChange = DataSource1DataChange
  end
  object Table1: TTable
    Active = True
    AfterInsert = Table1AfterInsert
    BeforePost = Table1BeforePost
    DatabaseName = 'DBDEMOS'
    TableName = 'COUNTRY.DB'
    // 5 field objects omitted
  end
end
```

Kao što možete videti iz prethodnog listinga, program sadrži nekoliko obrada događaja koje nismo koristili u prethodnim aplikacijama, koje su koristile kontrole koje prepoznaju podatke. Prvo, moramo prikazati podatke trenutnog sloga u vizuelnim komponentama (kao na slici 9.13), obradom `OnDataChange` događaja komponente `DataSource1`:

```
procedure TForm1.DataSource1DataChange(Sender: TObject; Field: TField);
begin
  EditName.Text := Table1Name.AsString;
  EditCapital.Text := Table1Capital .AsString;
  ComboContinent.Text := Table1Continent.AsString;
  EditArea.Text := Table1Area.AsString;
  EditPopulation.Text := Table1Population.AsString;
end;
```



SLIKA 9.13 *Izlaz primera NonAware u modu Browse. Program ručno pronalazi podatke svaki put kada se slog izmeni.*

Obarada `OnStateChange` događaja kontrole koristi isti kod koji smo videli u primeru GridDemo. Ovoga puta status tabele se prikazuje na stautsnoj liniji. Kada korisnik počne da unosi podatke u jedno od polja za izmene, ili prikaže listu combo polja, program prevodi tabelu u mod za izmene:

```
procedure TForm1.EditKeyPress (Sender TObject; var Key: Char);
begin
  if not (Table1.State in [dsEdit, dslnsertl]) then
    Table1.Edit;
end;
```

Ovaj metod je povezan sa `OnKeyPress` događajem pet komponenata i sličan je obradi događaja `OnDropDown` za combo polje. Kada korisnik napusti vizuelne kontrole, obrada `OnExit` događaja kopira podatke u odgovarajuća polja kao u ovom slučaju:

```
procedure TForm1.EditCapitalExit(Sender: TObject);
begin
  if (Tablel.State = dsEdit) or (Tablet.State dslnsert) then
    Table1Capital.AsString := EditCapitli Text;
end;
```

Operacija se obavlja samo ukoliko se tabela nalazi u modu Edit, to jest, samo ukoliko korisnik unosi podatke u ovu ili neku drugu kontrolu. Ovo zapravo nije idelano, jer se obavljaju dodatne operacije čak i ako se ne promeni tekst polja, međutim, dodatni koraci se dovoljno brzo izvršavaju te se o njima ne treba brinuti. Za prvo polje za izmene proveravamo tekst pre nego što ga kopiramo, pozivajući izuzetak ukoliko je polje prazno:

```
procedure TForm1.EditNameExit(Sender: TObject);
begin
  if (Table1.State = dsEdit) or (Table1.State = dsInsert) then
    if EditName.Text <> '' then
      Table1Name.AsString := EditNameText
    else
    begin
      EditName. SetFocus;
      raise Exception Create ('Undefined Country');
    end;
end;
```

Alternativni pristup za testiranje vrednosti polja je obrada `BeforePost` događaja skupa podataka (efekat je prikazan na slici 9.14). Imajte na umu da se operacija prosleđivanja u ovom primeru ne obrađuje specifičnom kontrolom, već se obavlja čim korisnik pređe na drugi slog ili umetne novi:

```
procedure TForm1.Table1BeforePost (DataSet: TDataSet);
begin
  if Table1Area Value < 100 then
    raise Exception.Create ('Area too small');
end;
```

SLIKA 9.14 *Poruka o grešci prikazana kada je vrednost za površinu previše mala*

U svakom od ovih slučajeva, alternativa izuzetku je određivanje vrednosti. Ipak, ukoliko je za polje određena podrazumevana vrednost, bolje ju je što pre odrediti tako da korisnik može videti koja će se vrednost poslati bazi podataka. Da biste ovo postigli, možete obraditi `AfterInsert` događaj skupa podataka, koji se inicira odmah pošto se kreira novi slog (takođe smo mogli da upotrebimo događaj `OnNewRecord`):

```
procedure TForm1.Table1AfterInsert (DataSet: TDataSet);
begin
  Table1Continet.Value := 'Asia';
end;
```

Slanje zahteva bazi podataka

Možete još više prilagoditi korisnički interfejs Vaše aplikacije ukoliko odlučite da ne koristite isti skup operacija izmena kao standardne Delphi kontrole koje prepoznaju podatke. Na ovaj način ćete imati potpunu slobodu, mada se mogu javiti sporedni efekti (recimo ograničena mogućnost konkurentnosti, a to ću razmatrati u narednom poglavlju).

Za ovaj novi primer sam promenio prvo polje za izmene u combo polje, i promenio sam sve kontrole koje su se odnosile na operacije nad tabelom (koje su odgovarale DBNavigator kontrolama) sopstvenim kontrolama koje se koriste za dobijanje podataka iz baze podataka i za ažuriranje. Da bih istakao razlike, ja sam čak uklonio komponentu DataSource.

Metod `GetData`, koji je povezan sa odgovarajućom kontrolom, preuzima polja koja odgovaraju slogu koji je naznačen prvim combo poljem:

```
procedure TForm1.GetData;
begin
Table1.FindNearest ([ComboName.Text]);
  ComboName.Text := Table1Name.AsString;
  EditCapital Text := Table1Capital .AsString;
  ComboContinent.Text := Table1Continent.AsString;
  EditArea.Text := Table1Area.AsString;
  EditPopulation.Text := Table1Population.AsString;
end;
```

Ovaj metod se poziva svaki put kada korisnik klikne kontrolu, odabere element combo polja, ili pritisne taster Enter kada se nalazi u combo polju:

```
procedure TForm1.CamboNameClick(Sender: TObject);
begin
  GetData;
end;

procedure TForm1.CamboNameKeyPress(Sender: TObject; var Key: Char);
begin
  if Key = #13 then
    GetData;
end;
```

Da bi primer dobro funkcionisao, combo polje se prilikom pokretanja popunjava nazivima svih zemalja koje se nalaze u tabeli:

```
procedure TForm1. FormCreate(Sender: TObject);
begin
  // fill the list of names
  Table1.Open;
  while not Table1.Eof do
  begin
    ComboName.Items.Add (Table1Name.AsString);
    Table1.Next;
  end;
end;
```

Ovakvim pristupom combo polje postaje nekakva vrsta selektora slogova, kao što možete videti na slici 9.15. Primetićete da zahvaljujući ovoj selekciji programu nisu potrebne kontrole za navigaciju.

SLIKA 9.15 *U primeru SendToDb, u combo polju možete selektovati slog koji želite da prikažete*

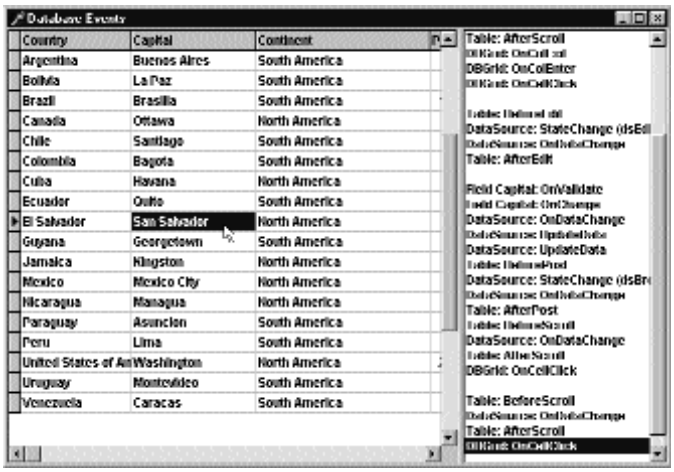
Konačno, korisnik može promeniti vrednosti kontrole i kliknuti kontrolu Send. Kod koji treba da se izvrši, zavisi od toga da li je to operacija ažuriranja ili umetanja. Ovo možemo odrediti ukoliko proverimo naziv (mada se ovim programom pogrešno unet naziv više ne može ispraviti):

```
procedure TForm1.SendData;
begin
// raise an exception if there is no name
  if ComboName.Text = ' ' then
    raise Exception.Create ('Insert the name');

  // check if the record is already in the table
  if Table1.FindKey ([ComboName.Text]) then
  begin
    // modify found record
    Table1. Edit;
    Table1Capital .AsString := EditCapital Text;
    Table1Continent.AsString := ComboContinent.Text;
    Table1Area.AsString := EditArea.Text;
    Table1Population.AsString := EditPopulation.Text;
    Table1.Post;
  end
  else
  begin
    // insert new record
    Table1.InsertRecord ([ComboName.Text,
      EditCapital Text, ComboContinent.Text,
      EditArea.Text, EditPopulation.Text]);
    // add to list
    ComboName.Items.Add (ComboName Text)
end;
```

Pre nego što se podaci pošalju tabeli, možete izvršiti bilo kakvu proveru valjanosti podataka. U ovom slučaju nema mnogo smisla obrađivati događaje komponenata baze podataka jer imamo punu kontrolu nad operacijama ažuriranja ili umetanja.

Događaji baze podataka

Da bih ilustrovao kako možete koristiti događaje aplikacije za baze podataka, ja sam napisao jednostavan program koji beleži sve događaje koji su se odigrali. Ovaj program obrađuje sve događaje komponenata tabele i izvora podataka (mada se neki od ovih događaja zapravo neće izvršiti ukoliko ne dodate kod koji ću kasnije opisati). Za svaki događaj sam samo poslao njegov opis u listu, a efekat možete videti na slici 9.16.



SLIKA 9.16 *Izlaz programa DbEvts koji beleži sve događaje koji su vezani za komponente baze podataka*

Većina obrada događaja samo prikazuje naziv komponente i događaja, kao u

```
procedure TForm1.Table1AfterEdit (DataSet: TDataset);
begin
  AddToList ('Table: AfterEdit');
end;
```

Događaji polja su nešto složeniji, ali koriste jednu obradu događaja za razne komponente polja:

```
procedure TForm1.FieldChange (Sender: TField);
begin
  AddToList ('Field' + Sender.FieldName + ': OnChange');
end;
```

Metod `AddToList` formulara dodaje novi element listi i selektuje ga, automatski skrolujući listu ukoliko je to potrebno:

```
procedure TForm1.AddToList (Str: string);
begin
  // add item and select it
  Listbox1.ItemIndex := Listbox1.Items.Add (Str);
end;
```

Konačno, program sadrži kontekst meni koji je povezan sa listom i služi za uklanjanje elemenata iz liste ili njihovo čuvanje u fajlu. Meni, takođe, sadrži komandu kojom možete dodati praznu

liniju i time odvajati blokove događaja. Ova operacija se takođe automatski obavlja pomoću tajmera koji dodaje praznu liniju listi, izuzev ukoliko je poslednji element takođe prazan string. Na ovaj način izlaz je čitljiviji, kao što možete videti na slici 9.16.

Veoma je važno proučiti izlaz ovog programa kao i njegov kod. Možete probati da izvršite razne operacije nad tabelom koristeći DBGrid, kao što su umetanje, izmene ili uklanjanje slogova, i videti odgovarajući efekat u terminima događaja VCL komponenata. Da biste videli više događaja, možete odrediti vrednost `True` za svojstvo `Filtered`, definisati polje koje se izračunava, pokušati da izazovete greške (na primer, da duplirate vrednost polja u kome se nalaze nazivi), dodate polje za potvrdu za otvaranje ili zatvaranje tabele i tako dalje.

Događaji polja

Program DbEvts prikazuje pozive događaja `OnChange` i `OnValidate` objekata polja. Druga dva događaja, `OnSetText` i `OnGetText`, se ne prikazuju jer se obrade ovih događaja ne pozivaju da bi se naznačilo da se desila operacija. Nasuprot tome, njihove obrade moraju izvršiti operaciju preuzimanja ili određivanja podataka za odgovarajuće objekte polja.

Ova dva događaja su prilično specifična, a njihova upotreba nije tako jednostavna kao što se to čini na prvi pogled. Zbog toga je za njih potreban poseban primer nazvan FldText. Ovo je samo revizija primera DbAware koji je opisan ranije u ovom poglavlju, a ovde je kontrola DBRadioGroup zamenjena kontrolom DBListBox. Problem je u tome što se DBListBox povezuje direktno sa stringom polja, dok ja želim da je povežem sa celobrojnim poljem, pri čemu svaki broj predstavlja opciju. Naravno, ne želim da korisnik vidi ili selektuje broj, te moram da mapiram brojeve koji se čuvaju u bazi podataka stringovima koji se vide na ekranu. U ranijem primeru kontrola DBRAdioGroup je obezbeđivala takvo mapiranje. Sada moram da upotrebim drugačiji pristup.

U primeru FldTxt polje Department sadrži dve obrade događaja `OnGetText` i `OnsetText`. U obradi događaja `OnGetText` možete izdvojiti numeričku vrednost polja `Sender` i odrediti vrednost `Text` referentnog parametra:

```
procedure TDbaForm.Table1DepartmentGetText(Sender TField;
  var Text: String; DisplayText: Boolean);
begin
  case Sender.AsInteger of
    1: Text := 'Sales'
    2: Text := 'Accounting'
    3: Text := 'Production'
    4: Text := 'Management'
  else
    Text := '[Error]';
  end;
end;
```

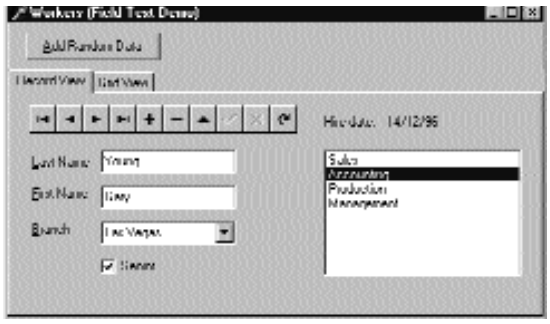
UPOZORENJE

U kodu obrade događaja `OnGetText` ne možete se referisati na tekst polja, na primer, koristeći svojstvo `DisplyText` ili metod `GetData` jer bi oni pozvali događaj `OnGetText`, što bi dovelo do beskonačne rekurzije. ■

U obradi `OnSetText` događaja možete proveriti string i odrediti vrednost polja, prema pravilima konverzije, u ovom slučaju jednostavnog mapiranja vrednosti koje se obavlja iskazom `if-then-else`:

```
procedure TDbaFormTable1DepartmentSetText(Sender TField;
  const Text: String);
begin
  if Text = 'Sales' then
    Sender.Value := 1
  else if Text = 'Accounting' then
    Sender.Value := 2
  else if Text = 'Production' then
    Sender.Value := 3
  else if Text = 'Management' then
    Sender.Value := 4
  else
    raise Exception.Create (
      'Error in Department field conversion');
end;
```

Ne samo da je vidljiva vrednost u DBListBoxu (kao što možete videti na slici 9.17), već se prikazuje i u DBGridu. Nasuprot tome, u primeru DbAware tabela je prikazivala numeričke vrednosti.



SLIKA 9.17 *Izlaz primera FldText koji prikazuje upotrebu događaja OnGetText i OnSetText objekata polja*

Promena datuma upotrebom kalendara

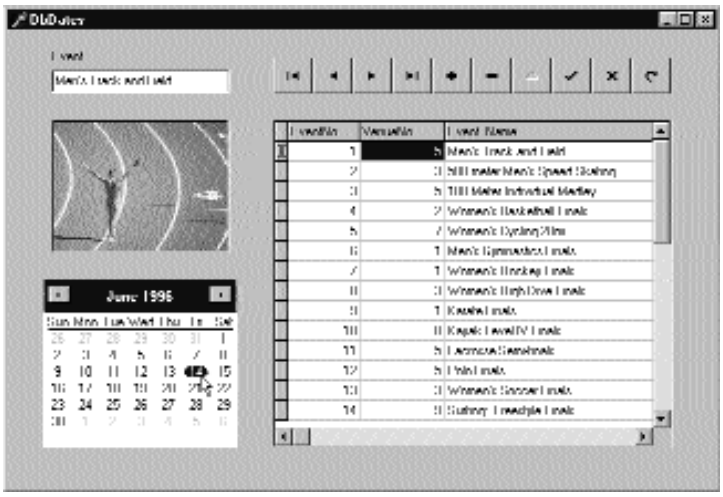
Kao poslednji primer upotrebe nevizuelnih kontrola, DbDate aplikacija pokazuje kako da upotrebite komponentu MonthCalendar za obradu datuma lepom grafičkom komponentom umesto običnim poljem za izmene. Ovaj primer je baziran na tabeli Events baze podataka DBDemos koja prikazuje olimpijske događaje.

Ovaj primer koristi (prvi put) kontrolu DBImage, sa sledećim vrednostima (čiji je efekat ilustrovan slikom 9.18):

```
object DBImage1: TDBImage
  DataField = 'Event_Photo'
  DataField = DataSource1
  Stretch = True
end
```

NAPOMENA

Grafička, Memo i BLOB polja se u Delphiju obrađuju kao i ostala polja. Samo povežite odgovarajući editor i većinu posla obavlja sistem u pozadini. ■



SLIKA 9.18 *Odabiranje datuma upotrebom kalendara*

Mada kontrola DBImage funkcioniše bez mnogo napora sa Vaše strane, ipak moramo povezati kontrolu MonthCalendar sa odgovarajućim poljem, obradom dva događaja DataSource kontrole:

```
procedure TForm1.DataSource!DataChange (Sender: TObject; Field: TField);
begin
  MonthCalendar1.Date := Table1Event_Date.Value;
end;

procedure TForm1.DataSource1UpdateData (Sender: TObject);
begin
  Table1Event_Date.Value := MonthCalendar1.Date;
end;
```

Pored kopiranja podataka prethodnim kodom, program takođe mora da prebaci tabelu u mod za izmene kada korisnik klikne kontrolu MonthCalendar. Najočigledniji pristup je da napišete obradu događaja `OnClick` kontrole MonthCalendar:

```
procedure TForm1.MonthCalendar1Click (Sender: TObject);
begin
  Table1.Edit;
end;
```

Ipak, ovaj kod ne funkcioniše pravilno. Kada tabelu prebacite u mod za izmene, događaj `OnDataChange` je izvršava još jednom, resetujući izbor u kalendaru. Sve u svemu, kada korisnik prvi put klikne kontrolu, ne menja se selekcija. Da bismo izbegli ovaj problem, možemo podesiti zastavicu `OnClick` događaja i testirati je u obradi događaja OnDataChange, ili možemo privremeno isključiti obradu drugog događaja. U narednom kodu sam primenio ovaj drugi pristup:

```
procedure TForm1.MonthCalendar1Click (Sender: TObject);
begin
  // disconect handler
  DataSource1.OnDataChange := nil;
  // set table in edit mode
  Table1.Edit;
  // reconnect handler
  DataSource1.OnDataChange := DataSource1DataChange;
end;
```

Pretraživanje tabela baze podataka

U našim dosadašnjim primerima uvek smo pristupali tabeli baze podataka određivanjem njenog naziva u vreme dizajniranja. Ali, šta se dešava ukoliko ne znate sa kojom tabelom treba povezati program? Na prvi pogled možete pomisliti da ukoliko ne znate detalje baze podataka u vreme dizajniranja, nećete moći da kreirate formulare i operišete sa tabelom. To nije tačno. Određivanje svega u vreme dizajniranja je sigurno lakše. Promena gotovo svega u vreme izvršavanja zahteva mnogo više kodiranja. To je ono što sam ja učinio u narednom primeru nazvanom Tables koji prikazuje kako treba pristupiti spisku baza podataka koje Borland Database Engine može da otvori, kako pristupiti tabelama svake od baza podataka i kako odrediti koja polja treba prikazati.

Izbor baze podataka i tabele u vreme izvršavanja

Za primer Tables sam pripremio formular sa combo poljem koje možete upotrebiti za selektovanje baze podataka, i listu koju možete upotrebiti za selektovanje tabele te baze podataka. Formular takođe sadrži DBGrid koji se može povezati sa odabranom tabelom baze podataka. Izlaz ovog programa možete videti na slici 9.19.

Kada se program pokrene, popunjava se combo polje, popunjava se lista (prema prvom elementu combo polja), a zatim se prikazuje tabela u DBGrid komponenti (simulirajući tako selektovanje prvog elementa liste):

```
procedure TMainForm.FormCreate (Sender: TObject);
begin
  Session.GetDatabaseNames (ComboBox1.Items);
  // force an initial list in the listbox
  ComboBox1.Text := 'DBDEMOS';
  ComboBox1Change (Self);
  // force an initial selection in the DBGrid
  ListBox1.ItemIndex := 0;
  ListBox1Click (Self);
end;
```

SLIKA 9.19 *Izlaz programa Tables koji prikazuje podatke tabele selektovane u vreme izvršavanja*

Ključni deo je poziv **procedure** `GetDatabaseNames` globalnog objekta `Session`. Objekat klase `TSession` se automatski definiše i inicijalizuje svakom Delphi aplikacijom za baze podataka (čak i kada Vi ne definišete ni jedan takav objekat), i da biste pristupili njegovim metodima, potrebno je samo da se referišete na DBTables jedinicu u iskazu `uses`. Kada se combo polje popuni, program odmah selektuje jednu od baza podataka i inicira obradu `ComboBox1Change` događaja u kome se koristi još jedan metod `TSession` klase, metod `GetTablesName`. Ovaj metod ima pet parametara: naziv baze podataka, string filtriranja, dve Boolean vrednosti koje označavaju da li je potrebno uključiti ekstenzije fajla (samo za lokalne tabele) i da li uključiti sistemske tabele (za SQL baze podataka), i `TStringsList` koji će biti popunjen nazivima tabela. Evo koda koji izvršava program kada korisnik odabere element combo polja:

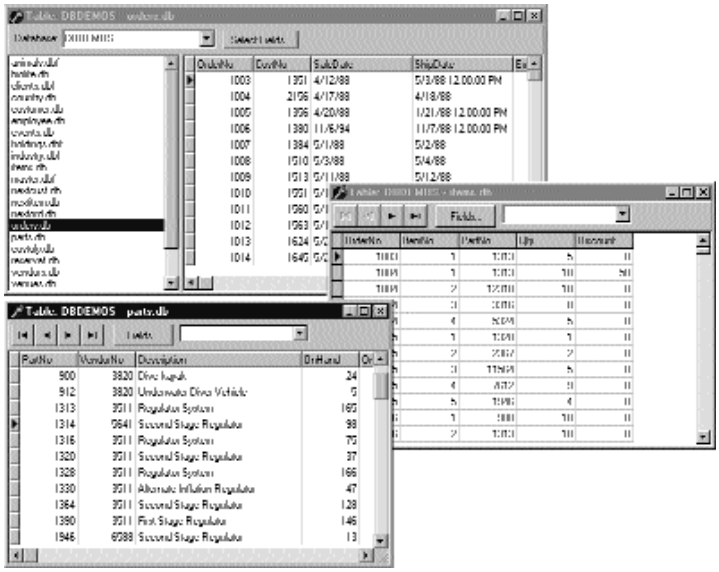
```
procedure TMainForm.ComboBox1Change (Sender: TObject);
begin
  Session.GetTableNames (ComboBox1.Text, ' ',
    True, False, ListBox1.Items);
end;
```

U metodu `FormCreate` sledeći korak se automatski izvršava prilikom pokretanja; program ispunjava DBGrid kao kada bi bio selektovan element liste:

```
procedure TMainForm.ListBox1Click (Sender: TObject);
begin
  Table1.Close;
  Table1.DatabaseName := ComboBox1.Text;
  Table1.Tablename := Listbox1.Items [Listbox1.ItemIndex];
  Table1.Open;
  Caption := Format ('Table: %s - %s',
    [Table1.DatabaseName, Table1.tablename]);
end;
```

Prikazivanje više tabela

Program omogućava korisniku da prikaže sadržaj bilo koje tabele. Kao dodatak, kada korisnik dva puta klikne listu, program prikazuje tabelu na zasebnom formularu. Ovim se omogućava korisniku da otvori više neprioritetnih formulara i prikaže više tabela istovremeno, kao što možete videti na slici 9.20.



SLIKA 9.20 *Program Tables se može upotrebiti za prikazivanje dveju ili više tabela*

Kada korisnik dva puta klikne listu glavnog formulara, kod kreira `TGridForm` objekat, povezuje komponentu `Table1` ovog formulara sa odgovarajućom bazom podataka i tabelom, i prikazuje formular:

```
procedure TMainForm.ListBox1DblClick(Sender: TObject);
var
  GridForm: TGridForm;
begin
  GridForm := TGridForm.Create (Self);
  {connect the table component to the selected
  table and activate it}
  GridForm.Table1.DatabaseName := ComboBox1.Text;
  GridForm.Table1.TableName :=
    Listbox1.Items [Listbox1.ItemIndex];
  try
    GridForm.Table1.Open;
    GridForm.Show;
  except
    GridForm.Close;
  end;
end;
```

SAVET

Primetićete da prethodni kod samo kreira formular, ali da ga nikada ne uklanja. Na formularu je da se sam ukloni u svom `OnClose` događaju određivanjem vrednosti `caFree` `Action` referentnog parametra. ■

Kada je kreiran sekundarni formular, program popunjava combo polje nazivima polja tabele. Ipak, ovaj kod ne može ići u događaj `OnCreate` formulara jer se formular kreira pre nego što se komponenta `Table1` pravilno podesi. Umesto da dodam svoj metod i da ga pozivam, ja sam koristio obradu `OnShow` događaja, koji takođe određuje zaglavlje formulara prema nazivu tabele i baze podataka:

```
procedure TGridForm.FormShow(Sender: TObject);
var
  I: Integer;
begin
  Caption := Format ('Table: %s - %s',
    [Table1.DatabaseName, Table1.TableName]);

  // fill the combo box with the names of the fields
  ComboBox1. Items.Clear;
  for I := 0 to Table1.FieldCount - 1 do
    ComboBox1.Items.Add (Table1.Fields[I] .FieldName);
end;
```

NAPOMENA

Moguće unapređenje ovog programa bi bilo generisanje formulara koji koristi kontrole koje prepoznaju podatke, a koje se biraju na osnovu tipa podataka polja. Videćete da Database Form Wizard može generisati formulare slične formularima na mom web sajtu, `www.marcocantu.com`. ■

Kakva je uloga combo polja? Svaki put kada korisnik selektuje element, odgovarajuće polje je ili prikazano ili sakriveno, već prema trenutnom stanju:

```
procedure TGridForm.ComboBox1Change (Sender: TObject);
begin
  // toggle the visibility of the field
  Table1.FieldByName (ComboBox1.Text).Visible) :=
    not Table1.FieldByName (ComboBox1.Text).Visible;
end;
```

Primetićete upotrebu metoda `FieldByName` kojim se dobija polje upotrebom trenutne selekcije combo polja i upotrebe svojstva `Visible`. Kada polje postane nevidljivo, odmah se uklanja iz komponente Grid koja je povezana sa tabelom. Zbog toga određivanjem ovog svojstva možemo automatski promeniti komponentu Grid.

Combo polje koje sam smestio na paletu alata GridForm dobro funkcioniše, ali ukoliko je potrebno da seletujete nekoliko polja iz velike tabele, veoma je sporo i sklono greškama. Kao alterantivu sam kreirao formular za izmenu polja, koji koriste kako glavni formular tako i sekundarni formulari. To je treći formular primera Tables koji je nazvan FieldsForm.

Ovaj formular se prikazuje kao prioritetni okvir za dijalog, tako da svaki put možemo koristiti jedan globalni objekat. Novi formular nema sopstveni kod. Kada se formular aktivira, njegova

lista se ispunjava nazivima polja tabele. Istovremeno, kodom se selektuju elementi liste koji odgovaraju vidljivim poljima, kao što možete videti na slici 9.21.

SLIKA 9.21 *Kada se formular aktivira, njegova lista se ispunjava nazivima polja tabele*

Korisnik može promeniti selekciju svakog elementa liste dok je aktivan prioritetni formular. Kada se zatvori, drugi formular dobija vrednosti selektovanih elemenata i prema njima se određuje vrednost svojstva polja Visible. Evo kompletnog koda ovog metoda:

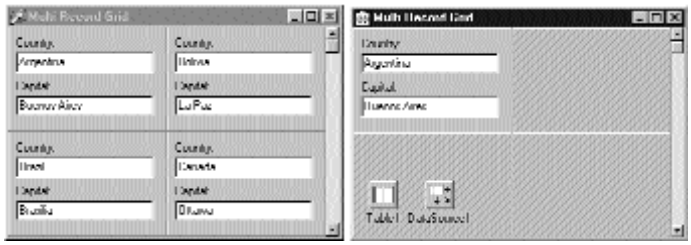
```
procedure TGridForm.SpeedButton1Click(Sender: TObject);
var
  I: Integer;
begin
  FieldsForm.FieldsList.Clear;
  for I := 0 to Table1.FieldCount - 1 do
  begin
    FieldsForm.FieldsList.Items.Add (
      Table1.Fields [I] .FieldName);
    if Table1.Fields [I].Visible then
      FieldsForm. FieldsList.Selected [I] True;
  end;
  if FieldsForm.ShowModal = mrOK then
    for I := 0 to Table1.FieldCount - 1 do
      Table1.Fields.Visible [I]
        FieldsForm. FieldsList.Selected [I];
  FieldsForm. FieldsList.Clear;
end;
```

Ovim kodom se završava opis primera. Videli smo da možete da napišete aplikacije za baze podataka tako da se veći deo posla može obaviti u vreme izvršavanja, mada je ovakav pristup nešto složeniji.

Grid sa više slogova

Do sada smo videli da Grid možete koristiti za prikazivanje velikog broja slogova tabele baze podataka ili za izradu formulara sa specifičnim kontrolama koje prepoznaju podatke za različita polja, slog po slog. Postoji i treća mogućnost: upotreba objekta sa više slogova (DBCtrlGrid) koji Vam omogućava da smestite mnogo kontrola koje prepoznaju podatke na malu površinu formulara i da automatski duplirate te kontrole za veliki broj slogova.

Evo šta možemo da uradimo da bismo izradili primer Multi1. Kreirajte novi prazan formular, na njega postavite komponentu Table i komponentu DataSource i povežite ih sa tabelom COUNTRY.DB. Sada smestite DBCtrlGrid na formular, podesite veličinu i broj kolona i redova, postavite dve komponente za izmene i povežite ih sa poljima tabele Name i Capital. Da biste smestili ove DBEdit komponente, možete otvoriti Fields editor i prevući polja na Grid. U vreme dizajniranja Vi radite sa aktivnim delom Grida (videti sliku 9.22, desna strana slike), a u vreme izvršavanja ove kontrole vidite replicirane više puta (videti sliku 9.22, leva strana slike).



SLIKA 9.22 *DBCtrlGrid primera Multi1 u vreme dizajniranja (desno) i u vreme izvršavanja (levo)*

Evo najvažnijih svojstava DBCtrlGrid objekta i drugih komponenata ovog primera:

```
object Form1: TForm1
  Caption = 'Multi Record Grid'
  object DBCtrlGrid1: TDBCtrlGrid
  ColCount = 2
  DataSource = DataSource].
  RowCount = 2
  object DBEdit1: TDBEdit
    DataField = 'Name'
    DataSource = DataSource1
    end
    object DBEdit2: TDBEdit. . .
  end
  object Table1: TTable
    Active = True
    DatabaseName = 'DBDEMOS'
    TableName = 'COUNTRY.DB'
  end
  object DataSource1: TDataSource
    DataSet = Table1
  end
end
```

Zapravo, Vi možete jednostavno odrediti broj kolona i redova. Zatim možete svaki put promeniti veličinu kontrole, a širina i visina svakog panela će biti određena prema tome. Ono što ne možete je automatsko poravnanje Grida sa klijent oblašću formulara.

Pomeranje panela Control Grida

Da bismo poboljšali poslednji primer, možemo promeniti veličinu Grida koristeći metod `FormResize`. Možemo napisati sledeći kod (u primeru Multi2):

```
procedure TForm1.FormResize(Sender: TObject);
begin
  DBCtrlGrid1.Height := ClientHeight — Panel1.Height;
  DBCtrlGrid1.Width := ClientWidth;
end;
```

Ovaj kod funkcioniše, ali to nije ono što sam želeo. Ja sam želeo da povećam broj panela, ne da ih uvećam. Da bismo ovo postigli, možemo definisati minimalnu visinu panela i izračunati koliko panela može stati na raspoloživu površinu svaki put kada se promeni veličina formulara. Na primer, ja sam dodao još jedan iskaz metodu `FormResize` u primeru Multi2, koji postaje

```
procedure TForm1.FormResize(Sender: TObject);
begin
  DBCtrlGrid1.RowCount :=
    (ClientHeight — Panel1.Height) div 100;
  DBCtrlGrid1.Height := ClientHeight — Panel1.Height;
  DBCtrlGrid1.Width := ClientWidth;
end;
```

Umesto da istu stvar uradim sa kolonama Control Grid komponente, ja sam panelu dodao komponentu TrackBar. Kada se pozicija TrackBara promeni (opseg je između 2 i 20), program određuje broj kolona za Control Grid i menja im veličinu. Zapravo, ukoliko samo odredite broj kolona, kolone će imati istu širinu kao i ranije. Evo koda za događaj OnChange TrackBara:

```
procedure TForm1.TrackBar1Change(Sender: TObject);
begin
  LabelCols.Caption := Format (
    '%d Columns', [TrackBar1.Position]);
  DBCtrlGrid1.ColCount := TrackBar1.Position;
  DBCtrlGrid1.Width := ClientWidth;
end;
```

Ovaj kod i metod `FormResize` Vam omogućavaju da promenite konfiguraciju Control Grida u vreme izvršavanja na više načina. Primer pune verzije formulara možete videti na slici 9.23.

SLIKA 9.23 *Izlaz primera Multi2 sa velikim brojem kolona*

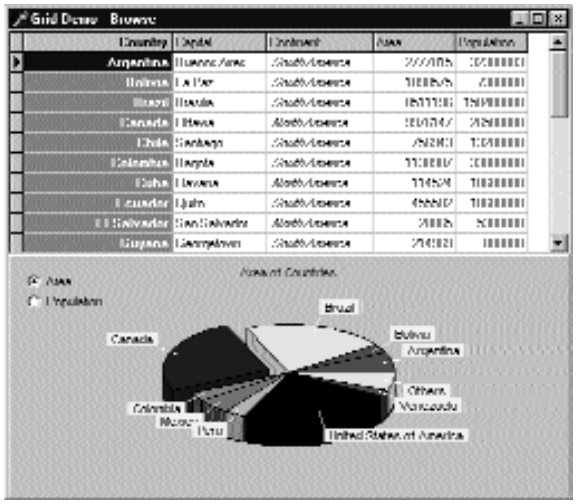
Grafikoni baze podataka

Još jedna interesantna komponenta koju možete upotrebiti u aplikacijama za baze podataka je verzija kontrole TeeChart koja prepoznaje podatke, koju je izradio David Berenda, a dobijate je uz Professional i Enterprise verzije Delphija. Ovu komponentu je veoma lako koristiti, naročito ukoliko Vaša verzija Delphija sadrži odgovarajući TeeChart Wizard (možete ga pronaći na Business strani File➥New okvira za dijalog).

Da bih prikazao upotrebu kontrole DBChart, ja sam ovu komponentu dodao primeru GridDemo. Nova aplikacija, nazvana ChartDB, prikazuje DBGrid u gornjem delu, a grafikon sa površina svake od zemalja u dnu, kao što se može videti na slici 9.24.

Program gotovo i da nema kod, jer se sva podešavanja mogu obaviti upotrebom odgovarajućeg editora, koji ima veliki broj opcija, ali ga je sasvim lako koristiti. Evo nekoliko ključnih svojstava komponenata, koje su preuzete iz opisa:

```
object DBChart1: TDBChart
  Legend.Visible = False
  Align = alClient
  object Series1: TPieSeries
    Marks.ArrowLength = 8
    Marks.Visible = True
    DataSource = Table1
    XLabelsSource = 'Name'
    ExplodeBiggest = 3
    OtherSlice.Style = poBelowPercent
    OtherSlice.Text = 'Others'
    OtherSlice.Value = 2
    PieValues.ValueSource = 'Area'
  end
end
```



SLIKA 9.24 *Izlaz primera ChartDB koji se zasniva na kontroli TDbChart*

NAPOMENA

Da biste razumeli ova svojstva i strukturu grafikona, možete pogledati primere Chart komponente u dodatnom poglavlju "Grafika u Delphiju" koje možete pronaći na adresi `www.sybex.com`. U tom poglavlju možete pronaći informacije o dinamičkom izvoženju sa web server aplikacije koja je proizvela grafikon, posle konvertovanja u JPEG sliku. ■

Ono što sam ja učinio, jeste da sam polje sa površinom prikazao kao izvor podataka za grafikon (svojstvo `PieValues.ValueSource`), upotrebio nazive polja za oznake (svojstvo `XLabelsSource`), i sve zemlje sa vrednošću od 2 procenta i manje prikazao sam jednim odeljkom označenim sa 'Others' (podsvojstva `OtherSlide`).

Dodao sam i dve opcione kontrole koje možete upotrebiti za izbor površine ili broja stanovnika. Kod za opcione kontrole samo određuje izvor podatka, pošto izvrši odgovarajuću konverziju tipova:

```
procedure TForm1.RadioPopulationClick (Sender: TObject);
begin
  DBChart1.Title.Text [0] := 'Population of Countries';
  (DBChart1.Series [0] as TPieSeries).
    PieValues.ValueSource := 'Population';
end;
```

Šta je sledeće?

U ovom poglavlju smo videli veliki broj primera pristupanja bazi podataka iz Delphi programa. Ja sam objasnio osnovne kontrole koje prepoznaju podatke, kao i programiranje aplikacija za baze podataka koje su zasnovane na standardnim kontrolama. Proučili smo internu arhitekturu objekata polja, kreirali smo potpuno nove tabele baze podataka u vreme dizajniranja i u vreme izvršavanja i uradili smo mnoge primere.

Pored toga što smo videli upotrebu kontrola koje prepoznaju podatke, koristili smo i nekoliko manuelnih pristupa. Možda se pitate kada pristup niskog nivoa ima smisla. Kratak odgovr je da koristite kontrole koje prepoznaju podatke, izuzev ukoliko nije potrebno da učinite nešto što nije uobičajeno i što se kosi sa unapred određenim ponašanjem kontrola koje prepoznaju podatke. Tipičan primer je upotreba takvih tehnika za ostvarivanje konkurentnosti u aplikacijama za više korisnika, što ćemo videti u naredna dva poglavlja.

Da li je to sve o Delphi programiranju baza podataka? Nikako. Delphi podrška baza podataka je veoma obimna i kompletna. Svrha ovog poglavlja je bila da Vam se da ideja šta sve možete učiniti, pri čemu je naglasak bio na Table komponenti za pristup bazi podataka. U narednom poglavlju ćemo usmeriti pažnju na Query komponentu, rad sa više tabela baze podataka (sa strukturama spajanja) i na mnoge druge napredne karakteristike. Takođe ćemo objasniti upotrebu novog Data Module Designera u Delphiju 5.

POGLAVLJE

10

Napredni pristup bazama podataka

U prethodnom poglavlju smo videli kako Delphi olakšava kreiranje aplikacija za baze podataka. Svi primeri programa su bili urađeni nad jednom tabelom i koristili su komponentu Table da bi se pristupilo podacima tabele. Sav kod koji je bio vezan za korisnički interfejs, bio je pomešan sa kodom za pristupanje bazi podataka; upotrebom modula podataka (data module) moći ćemo da odvojimo dve funkcionalne oblasti.

Ovo su samo neke teme kojima ćemo se baviti u ovom poglavlju koje je posvećeno nešto naprednijim tehnikama: Data Dictionary, BDE pozivi, spajanje tabela upotrebom SQL-a, master/detail povezivanje i lookup polja. Zatim ćemo se u poglavljima 11 i 12 posvetiti klijent/server programiranju i ADO komponentama.

Ovo poglavlje ćemo početi razmatranjem jedne od najvažnijih inovacija koje Delphi 5 nudi programerima baza podataka, a to je Designer modula podataka. Ovaj alat donosi novi vizuelni pristup struktuiranju aplikacija za baze podataka i značajno proširuje tradicionalne Delphi module podataka.

Delphi 5 Designer modula podataka

U prethodnom poglavlju smo koristili kontrole koje prepoznaju podatke i nevizuelne kontrole na formularu. To je bilo zgodno za jednostavan program, ali kada se korisnički interfejs, pristup podacima i model podataka nalaze u jednoj (često velikoj) jedinici, to je daleko od dobre ideje. Delphi je od verzije 2 koristio ideju modula podataka (data module), kontejnera nevizuelnih komponenata.

U vreme dizajniranja modul podataka je sličan formularu, ali u vreme izvršavanja postoji samo u memoriji. Klasa `TDataModule` je izvedena direktno iz klase `TComponent`, te nema nikakve veze sa Windows konceptom prozora. Za razliku od formulara, modul podataka sadrži samo nekoliko svojstava i događaja. Zbog toga se moduli podataka mogu posmatrati kao kontejneri komponenata i metoda u memoriji.

Ipak, moduli podataka su po mnogo čemu slični formularima. Kao i formular, modul podataka se odnosi na specifičnu Object Pascal jedinicu za definiciju svoje klase i za definiciju fajla formulara (DFM) koja sadrži spisak svih komponenata koje se nalaze u modulu i njihova svojstva. Evo dela koda iz DFM fajla modula podataka:

```
object DataModule2: TDataModule2
  Height = 159
  Width = 196
  object Table1: TTable . . .
  object DataSource1: TDataSource ...
end
```

Takođe, struktura Delphi jedinice modula podataka veoma je slična strukturi jedinice formulara. Ključna razlika je u roditeljskoj klasi:

```
type
  TDataModule2 = class (TDataModule)
```

Još jedna zajednička stvar između formulara i modula podataka je da se i jedni i drugi mogu kreirati kada se pokrene aplikacija, ili kasnije. Zapravo, moduli podataka se čak prikazuju u spisku na strani Forms okvira za dijalog Project Options.

Postoji nekoliko razloga za upotrebu modula podataka. Najjednostavniji je mogućnost da više formulara deli komponente za pristup podacima, kao što ću pokazati u primeru TwoViews. Ova tehnika funkcioniše uz upotrebu vizuelnog povezivanja formulara, mogućnosti pristupanja komponentama drugog formulara ili modula podataka u vreme dizajniranja (komandom File➥Use Unit). Drugi razlog je odvajanje podataka od korisničkog interfejsa, čime se poboljšava struktura aplikacije. Zapravo, Delphi Vam omogućava da još više proširite ovaj model do potpunog iskorišćenja sistema, upotrebom MIDAS tehnologije (koju ovlaš pominjem u Poglavlju 20).

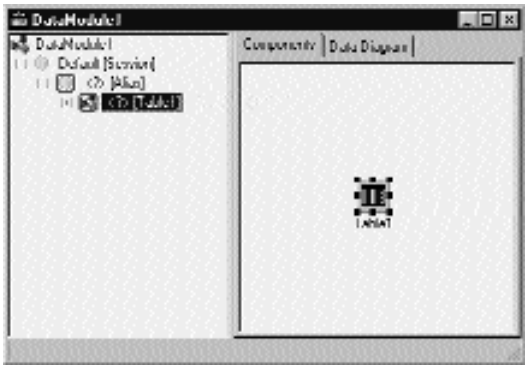
U Delphiju 5, Data Module Designer daje mnogo više razloga za upotrebu modula podataka. Možete odabrati komponente i povezati ih na lakši način, koristeći Tree View, a možete videti celokupnu strukturu aplikacije za bazu podataka (ili samo jednog njenog dela) pa čak i grafički povezati komponente i svojstva, koristeći pogled Data Diagram. Data Module Designer je proširenje modula podataka. Svaki put kada kreirate novi modul podataka (koristeći File➥New i izborom Data Module iz okvira za dijalog New Items), ili kada otvorite postojeći, prikazaće se novi dizajner.

Na levoj strani Data Module Designera se nalaze tri komponente kontejnera, organizovane po lokalnoj hijerarhiji. Na desnoj strani se nalaze dve strane kojma se prelazi u pogled Components ili Data Diagram, u zavisnosti od toga koju karticu ste odabrali. Pogled Components odgovara originalu modula podataka iz prethodnih verzija Delphija; u početku je to prazan prozor u koji možete da dodate komponente (ali ne i kontrole).

Pogled Tree

Pogled Tree Data Module Designera počinje jednim čvorom, samim modulom podataka. Vi možete odabrati komponentu iz palete (recimo Table) i mišem je prevući nad drvo. Dizajner će početi da logički organizuje informaciju dodajući pri tom dva dodatna čvora: BDE sesiju i alijas baze podataka, kao što možete videti na slici 10.1. Sesija je jednostavno osnovna sesija, koja predstavlja Delphi globalni objekat, a ne neku specifičnu komponentu. U nekim slučajevima možete je zameniti komponentom `TSession` koja ima specifična svojstva. Drugi čvor (pod sesijom) predstavlja alijas. Ne možete ga direktno editovati, ali ga možete promeniti dodeljivanjem vrednosti svojstvu `Database` tabele.

Ova dva čvora odgovaraju "lažnim" komponentama i zbog toga su njihove ikone prikazane sivom bojom. Sive ikone se koriste za komponente koje ne postoje u vreme dizajniranja. One su zaista komponente (u vreme dizajniranja i u vreme izvršavanja), ali kako su to unapred određeni objekti, koji se konstruišu u vreme izvršavanja a nemaju konstantne podatke koji se mogu menjati u vreme dizajniranja, Data Module Designer ne dozvoljava izmenu njihovih svojstava.



SLIKA 10.1 *Data Module Designer novog modula podataka pošto je dodata komponenta Table*

Postoje mnoge operacije koje možete obaviti unutar pogleda Tree. Na primer, posle određivanja alijasa možete prevući još jednu kontrolu pod alijas da biste ga direktno povezali sa bazom podataka. Na sličan način možete prevući komponentu izvora podataka ispod tabele i povezati ih. Ove operacije prevlačenja možete izvesti i nad komponentama koje se nalaze u pogledu Tree — na primer, možete promeniti skup podataka na koji referiše izvor podataka.

Primetićete da je ovo glavni aspekt novog dizajnera — oslobađa Vas posla ručnog povezivanja komponenata Session, Database, Table i DataSource upotrebom svojstava. Zapravo, zavisnosti koje vidite u drvetu se obično odnose na komponentu u "kontekstu roditelja", a određuju svojstvom. Sada možete odrediti roditeljski kontekst ili zavisnost kontejnera prevlačenjem elemenata na drvo. Na primer, Session komponente su kontekst u kojem jedna ili više Database komponenata funkcioniše, Tables i Queries funkcionišu sa kontekstom Database, a Field objekti postoje unutar Tables i Queries.

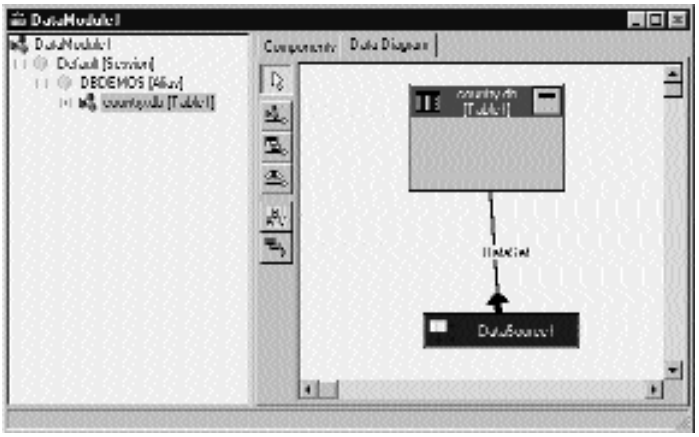
Kada desnim tasterom miša kliknete bilo koji element u pogledu Tree, prikazaće se kontekst meni koji je sličan meniju koji se prikazuje kada je komponenta formular (a u oba slučaja kontekst meni može sadržati elemente koji se odnose na editore komponenata). Možete čak i ukloniti elemente iz drveta; ukoliko element sadrži podelemente, Delphi će zatražiti potvrdu prilikom uklanjanja.

Pogled Data Diagram

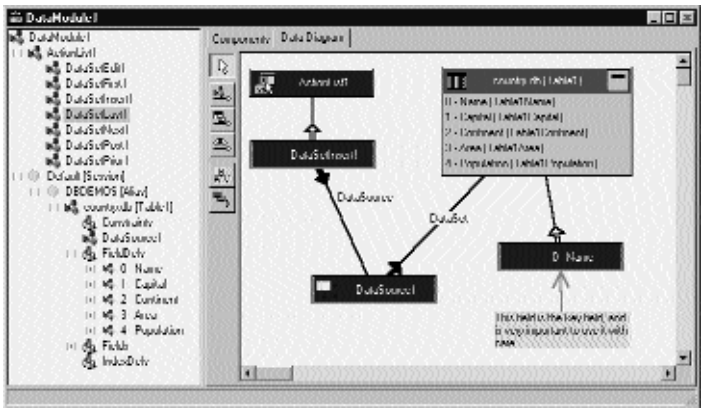
Ukoliko pogled Tree obezbeđuje nekoliko novih karakteristika u poređenju sa tradicionalnim modulom podataka, ono što je potpuno novo u Delphiju 5 jeste pogled Data Diagram. Ovaj pogled prikazuje zavisnosti među komponentama, uključujući master/detail zavisnosti, lookup veze, linkovana svojstva i generičke zavisnosti. Primetićete da, iako je to deo modula podataka, ovaj pogled nije ograničen na komponente koje su orijentisane na baze podataka; ovaj pogled se može koristiti za bilo koju nevizuelnu komponentu (menije, akcije i tako dalje).

Pogled Data Diagram se ne izrađuje automatski. Morate prevući komponente iz pogleda Tree u dijagram, mada su veze prikazane direktno ukoliko ste ih prethodno podesili. Ono što je dobro jeste da možete kreirati veze i odrediti svojstva jednostavnim iscrtavanjem strelica među komponentama. Na primer, posle premeštanja tabele i izvora podataka u pogled Data Diagram, možete da odaberete ikonu Property Selector, kliknete izvor podataka, prevučete pokazivač miša iznad tabele. Kada otpustite taster miša, dizajner će podesiti svojstvo zavisnosti, kao što možete videti na slici 10.2. Strelica će automatski prikazati naziv svojstva koje se koristi za povezivanje dveju komponenata, u ovom slučaju `Dataset`. Kao što možete videti, određivanje svojstava je *diskreciono*: ukoliko prevučete liniju svojstva zavisnosti od tabele do izvora podataka, na kraju ćete koristiti izvor podataka za svojstvo `MasterSource` tabele, povezujući dve komponente u suprotnom smeru.

To nije sve. Možete, takođe, prevući u pogled određena polja, i ona će biti povezana sa tabelom pomoću dete-zavisnosti, koja je označena belom strelicom. Možete čak premestiti ActionList i prevući ga u dijagram. Akcije će biti prikazane kao dete-elementi i, ukoliko se odnose na skup podataka, mogu biti povezane sa izvorom podataka. Da bih dobio sliku 10.3, ja sam izvršio ove korake, dodao sam komentar, dodao sam malo teksta i povezao sam komentar sa kom ponentom. Pored komponenata baza podataka Vi možete da upotrebite bilo koju nevizuelnu komponentu u Data Module Designeru, uključujući ActionList, menije, komponente Internet Producer i Dispatcher, MIDAS veze, Decision Cube komponente, pa čak i servere aplikacija.



SLIKA 10.2 *Svojstvo zavisnosti između dve komponente u pogledu Data Diagram*



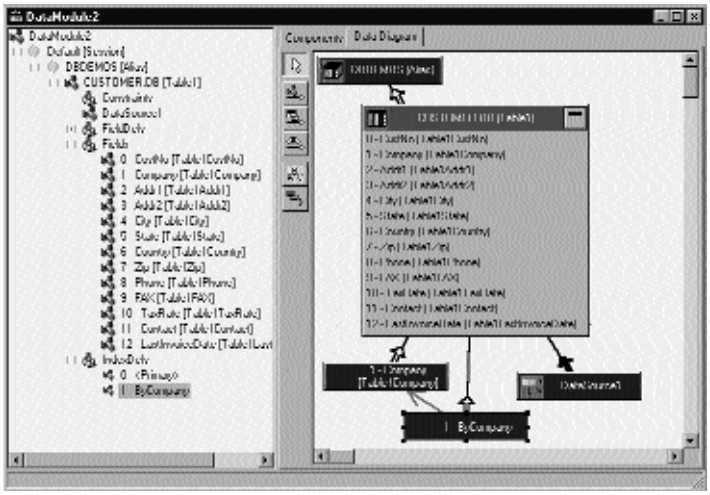
SLIKA 10.3 *Dijagram koji prikazuje složene zavisnosti između baze podataka i komponenata koje nisu vezane za bazu podataka (kao što su akcije)*

Mada možete upotrebiti pogled Data Diagram, obično u velikom prozoru, da biste podesili zavisnosti, njegova osnovna uloga je dokumentacija Vašeg dizajna. Zbog toga je veoma važno da se sadržaj ovog pogleda može odštampati, ali je takođe dobro da se može odštampati pogled Tree, naročito kada je previše veliki da stane na ekran. Da bi što više tabela stalo u dijagram, uvek možete prikazati ili sakriti spisak polja tabele tako što ćete kliknuti ikonu za umanjivanje/uvećavanje (minimize/maximize), koja se nalazi u gornjem desnom uglu polja.

Informacije u pogledu Data Diagram se čuvaju u zasebnom fajlu sa informacijama u vreme dizajniranja (DTI), ne kao deo DFM fajla. DTI fajlovi imaju strukturu koja je slična INI fajlovima, i očigledno nemaju nikakvu upotrebnu vrednost u vreme izvršavanja (nema nikakvog smisla uključiti ih u proces kompajliranja u izvršni fajl).

Modul podataka za više pogleda

Kao što sam ranije pomenuo, jedna od tradicionalnih upotreba modula podataka jeste obezbeđivanje više pogleda na iste podatke i za sinhronizovanje pogleda. To je ono što sam ja učinio u primeru TwoViews. Kasnije ću ovaj primer proširiti dodajući programu pravila podataka i mogućnosti filtriranja. U primeru TwoViews kreirao sam dva formulara i modul podataka. Modul podataka sadrži tabelu koja se odnosi na fajl CUSTOMER.DB baze podataka DBDEMOS i izvor podataka. Takođe sam kreirao `TField` komponente za svako od polja tabele. Rezultujući modul podataka možete videti na slici 10.4 (koja prikazuje konačnu strukturu modula podataka, uključujući polje i odgovarajući indeks).



SLIKA 10.4 *Modul podataka primera TwoViews*

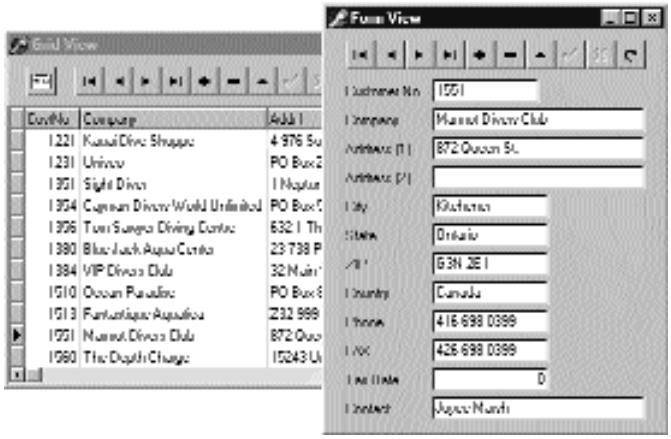
Ja sam, takođe, izradio paletu alata za glavni formular programa, koristeći panel poravnat sa vrhom formulara, točkić i DBNavigator. Točkić ima odgovarajuću ikonu i koristi se za prikazivanje sekundarnog formulara. Ostatak glavnog formulara je ispunjen DBGrid kontrolom. Posle povezivanja modula podataka sa formularom, komandom iz menija File➥Use Unit, možete podesiti svojstvo `DataSource` kako DBNavigatora tako i DBGrida u `DataModule2.DataSource1`.

SAVET

Pre nego što upotrebite još jednu jedinicu u formularu, potrebno je da pravilno imenujete jedinicu na koju želite da se referišete. Zapravo, ukoliko upotrebite jedinicu (na primer, Unit2), a zatim je preimenujete kada prvi put sačuvate fajl, veza će biti izgubljena i biće potrebno da ručno zamenite sve reference na preimenovanu jedinicu. Ovo se dešava čak i kad Delphi automatski umetne te uses iskaze, ili prilikom upotrebe komande File➥Use Unit. ■

Drugi pogled se zasniva na formularu koji sadrži mnogo DBEdit komponenata, po jednu za svako polje tabele baze podataka izuzev za poslednje polje. Umesto postavljanja velikog broja DBEdit komponenata i povezivanja svake od njih, možete otvoriti Fields editor komponente Table, dodati sva polja, selektovati sva polja izuzev poslednjeg, a zatim prevući selektovana polja

na sekundarni formular. Ovom jednostavnom operacijom Delphi je za Vas uredio sve odgovarajuće DBEdit i Labels komponente odjednom. Ja sam za svojstvo `Visible` sekundarnog formulara odabrao vrednost `True`, tako da odmah bude vidljiv kada se program pokrene, kao što možete videti na slici 10.5.



SLIKA 10.5 *Program TwoViews u vreme izvršavanja, sa dva sinhronizovana formulara koja prikazuju isti slog*

Ukoliko prikažete oba formulara, ona se održavaju sinhronizovano. Upotreba navigatora bilo kog formulara utiče na oba formulara. Zapravo, navigator nije povezan ni sa jednim formularom: povezan je sa izvorom podataka u modulu podataka, a sadržaj vizuelnih komponenata oba formulara se menja bilo kakvom izmenom komponenata za pristup podacima. Editujte jedan od formulara, a drugi će biti ažuriran čim se prihvate izmene. Dodajte novi slog, i akcija će se prikazati na oba formulara.

Primetićete da možete prikazivati slogove u vreme dizajniranja. Dok skrolujete tabelu, podaci sekundarnog formulara će se menjati, omogućavajući Vam da odredite veličinu DBEdit komponenata ukoliko je bilo koji slog preveliki da se može prikazati.

Određivanje svojstava polja i početnih vrednosti

Upotreba modula podataka za sinhronizovanje dva formulara je prilično zgodna i jednostavna. Mi želimo da dodamo programu još neke mogućnosti koje se odnose na same podatke, ne na specifične poglede. Na primer, u modulu podataka možemo izmeniti svojstva polja, upotrebom specijalne vrednosti za svojstva `EditMask` komponenata `Table1Phone` i `Table1FAX`. Ovo prilagođavanje će se odslikati na izlaz i editovanje ovih polja u oba formulara istovremeno.

Da bismo postigli nešto komplikovanije, mi možemo u tabelu uvesti pravilo, ili bar sugerisati korisniku. Želimo da automatski obezbedimo jedinstvenu vrednost oznake kupca i obezbedimo da bude jednaka najvećoj vrednosti plus jedan. Ovo sam postigao dodajući nešto koda modulu podataka.

U osnovi, mi želimo da odredimo pravilnu vrednost za polje Table1CustNo svaki put kada korisnik doda novi slog tabeli. Da bismo to postigli, možemo obraditi OnNewRecord događaj tabele na sledeći način:

```
procedure TDataModule2.Table1NewRecord (DataSet: TDataSet);
begin
  Table1CustNo.Value := Max + 1;
end;
```

Kako da izračunamo Max vrednost? Možemo jednostavno proći kroz tabelu, kao što smo to učinili u prethodnom poglavlju, i pronaći najveću vrednost polja CustNo. Ipak, to ne možemo učiniti u prethodnoj obradi događaja jer će to tabelu vratiti u mod tsBrowse iz moda tsInsert. Alternativa je izračunati najveću vrednost svaki put kada korisnik unese novi slog koristeći događaj BeforeInsert:

```
procedure TDataModule2.Table1BeforeInsert (DataSet: TDataSet);
begin
  ComputeMax;
end;
```

Procedura ComputeMax može pretražiti tabelu radi pronalaska najveće vrednosti, recimo, ovakvim kodom:

```
Max := 0;
try
  Table1.First;
  while not Table1.EOF do
  begin
    if Table1CustNo.AsInteger>Max then
      Max := Table1CustNo.AsInteger;
    Table1.Next;
  end;
```

Alternativa je dodavanje druge TTable komponente indeksirane na polje CustNo. Procedura ComputeMax bi tada jednostavno pogledala kraj tabele u potrazi za najvećom vrednošću polja CustNo:

```
procedure TDataModule2.ComputeMax;
begin
  Table2.Last;
  Max := Table2CustNo.AsInteger;
end;
```

Dodavanjem metoda modulu podataka približavamo se strukturi troslojne (three-tier) aplikacije. Ovaj kod je, zapravo, potpuno nezavisan od korisničkog interfejsa (dva pogleda). Kod ovog primera je veoma jednostavan, ali ističe ovu veoma važnu ideju.

UPOZORENJE

Alternativa prethodnom kodu, koji izračunava najveći identifikator koji se koristi u tabeli, jeste upotreba polja koje se automatski uvećava, bar kada koristite Paradox tabelu. Ovo je bolje rešenje u višekorisničkom okruženju u kome pristup koji je prikazan može dovesti do problema u slučaju dva konkurentna zahteva. ■

Standardno filtriranje tabele

Sada želimo da aplikaciji dodamo mogućnost filtriranja slogova u oba pogleda (ponovo koristeći modul podataka). Najjednostavnije filtriranje Delphi tabela je određivanje opsega vrednosti za indeksirano polje. Na primer, ja sam uredio tabelu programa TwoViews koristeći sekundarni indeks *ByCompany* (samo selektujte ovu vrednost za svojstvo `IndexName`). Zatim sam odabrao sve slogove između dve vrednosti, koje unosi korisnik, napisavši sledeće:

```
Table1.SetRange (['Abacus'], ['Custom']);
```

Alternativno, možete odrediti ključne vrednosti kao u metodu `GotoKey`, pozivajući redom metode `SetRangeStart`, `SetRangeEnd` i `ApplyRange`. Obično je mnogo lakše pozvati `SetRange` i proslediti dva niza vrednosti, sa onolikim brojem elemenata (i istim uređenjem) koliko ima polja u indeksu. Kada želite da prekinete filtriranje, jednostavno pozovite metod `CancelRange`.

Zapravo, u programu TwoViews ja nisam naznačio fiksirani opseg vrednosti kao što sam to prethodno nagovestio; umesto toga sam modulu podataka dodao metod `ChooseRange` koji pozivaju oba pogleda. Kako modul podataka nema vizuelni interfejs, on koristi okvir za dijalog preko koga korisnik unosi početnu i krajnju vrednost opsega. Mogao sam upotrebiti paletu alata u glavnom formularu umesto okvira za dijalog, ali sam želeo da povežem kod za određivanje opsega sa modulom podataka, a ne sa specifičnim formularom koji se koristi za prikazivanje podataka. Drugi pristup je dokiranje neprioritetnog dijaloga uz jednu stranu glavnog formulara, kao što je prikazano na slici 10.6. Na slici možete videti da sekundarni formular sadrži više komponenata, koje ćemo kasnije koristiti za prilagođavanje filtriranja tabele. Obratite pažnju na uticaj koji opseg ima na sadržaj tabele.

SLIKA 10.6 *Sekundarni formular koji se koristi za određivanje opsega i filtriranja tabele može se dokirati uz panel na nekoj od strana formulara*

Metod `ChooseRange` prikazuje formular `FormRange` (kao neprioritetni formular). Formular sadrži kontrolu Apply koju možete koristiti za aktiviranje novih vrednosti modula podataka:

```
procedure TFormRange.BitBtnClick (Sender: TObject);
begin
  with DataModule2.Table1 do
```

```
  begin
    if CheckBoxRange.Cecked then
      SetRange ([Edit1.Text], [Edit2.Text])
    else
      CancelRange;
  end;
end;
```

Korisničko filtriranje tabele

Pored određivanja opsega vrednosti za komponentu Table, možemo upotrebiti i sopstveni algoritam filtriranja. Jednostavno odaberite vrednost `True` za svojstvo `Filtering` Table komponente i za svaki slog će se pozivati događaj `OnFilterRecord`. U metodu koji je povezan sa ovim događajem, možemo odrediti korisnički filtar. Evo primera:

```
procedure TDataModule2.Table1FilterRecord (
  DataSet: TDataSet; var Accept: Boolean);
begin
  if (Table1Country.Value = 'US') or
    (Table1Country.Value = 'US Virgin Islands') or
    (Table1Country.Value = 'Jamaica') then
      Accept := True;
  else
    Accept := False;
end;
```

Ponovimo, povezivanje ovog pravila filtriranja sa modulom podataka uticaće na oba pogleda. Pored pisanja fiksiranog pravila, kao u prethodnom slučaju, mi korisniku možemo omogućiti da izgradi sopstveno pravilo sa komponentama koje su dodate donjem delu okvira za dijalog za opseg — drugim rečima, još jednim poljem za potvrdu i dvema listama, kao što je pokazano na slici 10.6. Liste su ispunjene nazivima država kada se formular kreira:

```
procedure TFormRange.FormCreate (Sender: TObject);
begin
  with DataModule2 do
  begin
  Table1. First;
  while not Table1.EOF do
  begin
    // add unigue values
    if not Table1Country.IsNull and
      (ListBoxCountries.Items.IndexOf (
        Table1Country.AsString) < 0) then
      ListBoxCountries.Items.Add (TablelCountry.Asstring);
    if not Table1State.IsNuIl and
       (ListBoxStates.Items.IndexOf (
          Table1State.AsString) < 0) then
        ListBoxStatesItems.Add (Table1State.AsString);
      Table1.Next;
    end;
    // reset the table
    Table1.First
  end;
end;
```

Ovaj kod proverava da li je vrednost trenutnog sloga Null ili da li već postoji u listi. Ukoliko ni jedno nije tačno, vrednost se dodaje odgovarajućoj listi. Ove dve liste bi trebalo ažurirati svaki put kada se doda novi slog tabeli baze podataka, ili svaki put kada se postojeći slog izmeni. Ja nisam dodao ovu mogućnost, ali bi trebalo da Vam bude sasvim jednostavno da je implementirate obradom `AfterUpdate` događaja tabele i pisanjem dveju linija koda, slično telu prethodne petlje `while`, pozivajući se na novi ili ažurirani slog.

Kada program popuni liste, one se prikazuju pored opcija opsega. Kontrola Apply takođe određuje filtre i osvežva tabelu:

```
with DataModule2.Table1 do
  begin
    ...
    Filtered := CheckBoxFiltering.Checked;
    Refresh;
```

Poziv `Refresh` je neophodan jer ukoliko se pravila promene kada je već aktivno filtriranje tabele, Delphi će automatski ponovo proračunati aktivne slogove. Najvažniji deo koda filtriranja je obrada `OnFilterRecords` događaja, koji proverava da li su zemlja ili država selektovani elementi dveju lista (koje omogućavaju višestruko selektovanje). Evo koda:

```
procedure TDataModule2 .Table1FilterRecord(
  DataSet: TDataSet; var Accept; Boolean);
begin
  {if the item corresponding to the country in the
  listbox is active, then view the record}
  with FormRange.ListBoxCountries do
    Accept := Selected [Items.IndexOf (Table1Country.AsString)];
  with FormRange.ListBoxStates do
    if Selected [Items.IndexOf (Table1Country.AsString)] then
      Accept := True;
end;
```

Primetićete da u drugoj `if` naredbi vrednost `Accept` treba da se doda prethodnoj iskazom `or`. Zapravo, mi možemo jednostavno odabrati `True` bez obzira na prethodnu vrednost (jer or uz `True` uvek daje `True`), ili prepustiti održavanje vrednosti (jer or uz `False` zadržava postojeću vrednost).

MDI aplikacija sa nezavisnim pogledima

Primer TwoViews je SDI aplikacija sa odvojenim pokretnim formularima na ekranu. To nije uvek najbolji korisnički interfejs; kao što smo razmatrali u Poglavlju 8, alternativa je upotreba MDI pristupa. Još jedno ograničenje programa do sada je da dva pogleda uvek prikazuju isti slog. Bilo bi lepo izraditi aplikaciju u kojoj bi pogledi sadržali različite aktivne slogove. Zato bi bilo dobro izraditi poglede koji prikazuju više slogova. To je ono što sam učino u narednom primeru, koji sam nazvao MdiView.

Kako je potrebno da imamo formulare koji prikazuju različit aktivan slog, možete doći u iskušenje da sasvim uklonite modul podataka i da na formular smestite komponente koje se odnose na baze podataka. To može dobro funkcionisati u jednostavnom primeru, ali u opštem slučaju je bolje zadržati logičko odvajanje i dizajniranje koje obezbeđuju moduli podataka. Alternativno rešenje je zadržati module podataka u programu i kreirati novu kopiju za svaku vezu formulara.

UPOZORENJE

Ukoliko u modul podataka smestite komponentu `Tdatabase`, ne možete kreirati više instanci modula podataka ukoliko ne odaberete vrednost `True` za svojstvo `HandleShared`. Ukoliko to ne učinite, program će generisati izuzetak "Name not unique in this context" (naziv nije jedinstven), jer dve komponente baze podataka u istoj sesiji baze podataka ne mogu imati isti naziv. ■

Glavni formular MdiView aplikacije za svojstvo `FormStyle` ima vrdnost `fsMdiFrame`. To je jedini formular koji se kreira prilikom pokretanja aplikacije, te se prikazuje prazan. Dete-formulari se kreiraju upotrebom komandi menija File i automatski kreiraju module podataka. Zbog toga sam ne samo uklonio dete-formulare i module podataka iz liste automatski kreiranih formulara u opcijama projekta, već sam čak uklonio i globalne promenljive koje se odnose na ove objekte.

SAVET

Ponoviću, uklanjanje globalnih promenljivih formulara koji imaju više instanci je, uopšte uzev, dobra ideja, da ne biste zamenili jednu instancu formulara (na koju se referiše globalna promenljiva) samom klasom. ■

Kod za kreiranje dete-formulara je prilično jednostavan. Nije potrebno voditi računa o kreiranim formularima, jer Windows MDI podrška to automatski čini za Vas:

```
procedure TFrameForm.NewRecordView1Click (Sender: TObject);
begin
  with TRecordForm.Create (Application) do
    Show;
end;
```

Kada se kreira pogled, on generiše modul podataka i povezuje ga sa lokalnim `DM` poljem, koje je deklarisano unutar formulara. Program zatim povezuje sve kontrole koje prepoznaju podatke sa izvorom podataka novokreiranog modula podataka:

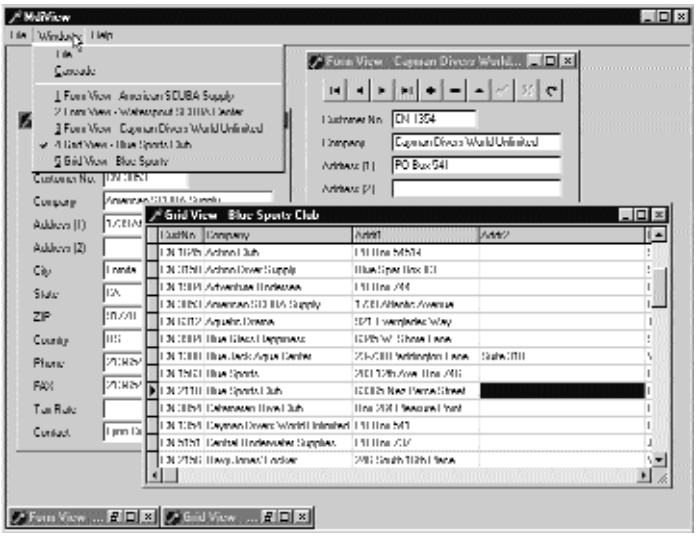
```
procedure TRecordForm.FormCreate (Sender: TObject);
var
  I: Integer;
begin
  DM := TCustomerDM.Create (self);
    // connect the navigator
  DBNavigator1.DataSource := DM.DataSource1;
    // connect all DBEdit controls
  for I := 0 to ControlCount - 1 do
    if Controls [I] is TDBEdit then
      TDBEdit (Controls [I]).DataSource :=
        DM.DataSource1;
end;
```

Jedini preostali deo koda za dva dete-formulara je kod za zatvaranje formulara, koji uklanja formular, određujući vrednost `caFree` za parametar `Action` događaja `OnClose`.

Modul podataka gotovo da i ne sadrži kod, jer sam ja uklonio izračunavanje maksimalnog ID-a koje se koristilo u prethodnim primerima ovog poglavlja. Jedina operacija koju obavlja modul podataka jeste promena naslova povezanog formulara da bi se odslikao aktivni slog. Ovo je korsno, jer lista MDI dete-prozora postaje besmislena ukoliko svi prozori istog tipa imaju isti naslov. Da bih ovo postigao, ja sam upotrebio trik — čuvanje svojstva `Hint` svakog formulara (koje se ne koristi) kao prvi deo naslova, za kojim sledi polje Company aktuelnog sloga:

```
procedure TCustomerDM.DataSource1DataChange(
  Sender: TObject; Field: TField);
begin
  (Owner as TForm).Caption :=
    (Owner as TForm).Hint + ' - ' + Table1.Company.AsString;
end;
```

Primer izlaza ovog programa možete videti na slici 10.7, kada je otvoreno nekoliko dete-prozora, a nekoliko je minimizovano. Primetićete da se različiti naslovi, koje smo odredili za formulare, prikazuju u Windows meniju; ovo se automatski obavlja MDI podrškom.



SLIKA 10.7 *Izlaz programa MdiView koji povezuje različite objekte modula podataka sa svakim pogledom*

Upotreba upita

Svi primeri sa bazama podataka do sada su koristili komponentu Table. Naredni primer pristupa podacima upotrebom komponente Query, a nazvan je DynQuery — ili dinamički upit. Ja sam upit povezao sa uobičajenim alijasom baze podataka `DBDEMO` i uneo tekst jednostavnog SQL iskaza:

```
select * from Country
```

Jednostavno aktivirajte komponentu Query i vrednosti polja prvog sloga bi trebalo da se pojave u poljima za izmene kao i obično. Naravno, ovo se dešava samo ukoliko je SQL iskaz koji ste umetnuli korektan. U suprotnom, Delphi će prikazati poruku o grešci, a upit neće biti aktiviran. Ukoliko želite da izmenite SQL iskaz upita u vreme izvršavanja, potrebno je da odredite vrednost `False` za svojstvo `Active`. Kada promenite tekst upita, možete ga ponovo aktivirati.

```
    Query1.Sql.Add ('select * from Country');
    Query1.Sql.Add ( 'where ' + Edit1.Text);
  end;
  try
    Query1.Open;
  except
    on EDatabaseError do
      ShowMessage ('Invalid condition:' #13 + Edit1.Text);
  end;
end;
```

Ovaj kod se izvršava svaki put kada polje za izmenu nije prazno. Kod sadrži test da bi se proverilo da li je tekst korektan SQL iskaz, a ukoliko nije, prikazuje poruku o grešci. Da bismo malo poboljšali program, poslednja opciona kontrola je automatski onemogućena svaki put kada je polje za izmenu prazno. Ova provera se obavlja u događaju OnChange komponente Edit.

Kada korisnik pritisne taster Enter dok se nalazi u polju za izmene, novi uslov se automatski aktivira, bilo da korisnik klikne opcionu kontrolu (vizuelno naznači selekciju i pozove obradu događaja) bilo pozivom obrade (jer selektovanje kontrole koja je već aktivna ne poziva obradu događaja):

```
procedure TQueryForm.Edit1KeyPress (Sender: TObject; var Key: Char);
begin
  if Key = #13 then
  begin
    if RadioButton4.Checked then
      RadioButton4Click (self)
    else
      RadioButton4.Checked := True;
    Key := #0;
  end;
end;
```

SAVET

Metod `Edit1KeyPress` pretvara taster Enter u taster null, da bi se izbegao unapred određeni zvuk kada pritisnete taster Enter u polju za izmene. ■

Kada pokrenete ovaj program, možete odabrati bilo koju od četiri kontrole, a efekat će se odmah videti na listi slogova u DBGrid kontroli. Na slici 10.8 su prikazana dva primera prilagođavanja SQL upita u vreme izvršavanja. Jedan primer prikazuje samo jednu državu, dok drugi prikazuje rezultat selektovanja prema opsegu stanovništva.

NAPOMENA

Po definiciji, Vi ne možete izmeniti rezultat upita. Da biste to mogli da učinite, prvo morate da odredite vrednost `True` za svojstvo `RequestLive` Query komponente. Na ovaj način podaci se mogu menjati, izuzev kada su ispunjeni dati uslovi određeni BDE-om. Jednostavni upiti koji se odnose na jednu tabelu baze podataka mogu biti "živi", dok složeni upiti koji spajaju nekoliko tabela to ne mogu biti. Komponenta `TUpdateSQL` i keširana tehnologija ažuriranja Vam omogućavaju da složene upite učinite "živim", kao što ćemo to videti u narednom poglavlju. ■

NAPOMENA

U Enterprise verziji Delphija, upite možete izraditi upotrebom grafičkog dela programa za izradu upita koji se zove SQL Builder. Ovo je dvosmerni alat koji može interpretirati tekstualne upite i prikazati ih grafički. Primere upotrebe SQL Buldera možete videti u Poglavlju 11. ■

Naravno, ovaj primer neće biti naročito interesantan. Zbog čega upotrebiti komponentu Query umesto komponente Table ukoliko samo želimo da prikažemo celu tabelu? Možemo iskoristiti komponentu Query dodavanjem opcionih kontrola za selektovanje različitih upita u vreme izvršavanja. Ja sam odlučio da dodam četiri različite opcije.

Prva opciona kontrola se koristi za izbor unapred određenog SQL iskaza i potvrđena je prilikom pokretanja. Druga i treća opciona kontrola se mogu koristiti za prikazivanje slogova sa određenim vrednostima za polje Continet, dodajući klauzulu `where` SQL iskazu. Poslednja opciona kontrola omogućava korisniku da unese tekst klauzule `where` SQL iskaza u polje za unos teksta.

Omogućavanje korisniku da unese SQL iskaz je opasno jer unošenje pogrešnog teksta može dovesti do greške, ali Delphi podrška obradi izuzetaka nam može pomoći da prevaziđemo rizik. Evo koda koji odgovara prvoj opcionoj kontroli:

```
procedure TNavigForm.RadioButton1Click (Sender: TObject);
begin
  Query1.Close;
  Query1.Sql.Clear;
  Query1.Sql.Add ('select * from Country');
  Query1.Open;
end;
```

Primetićete da svojstvo `SQL` nije string već lista stringova. Ovo se može upotrebiti za izradu veoma dugačkih upita (limit teksta za niz stringova je veoma veliki) i za definisanje različitih delova upita na različitim mestima u kodu i njihovo spajanje. Druga i treća opciona kontrla dele isti kod koji koristi njihovo svojstvo `Caption` prilikom izrade SQL iskaza:

```
Query1.Sql.Clear;
Query1.Sql.Add ('select * from Country');
Query1.Sql.Add ('where Continet = '' +
  (Sender as TRadioButton).Caption + '');
```

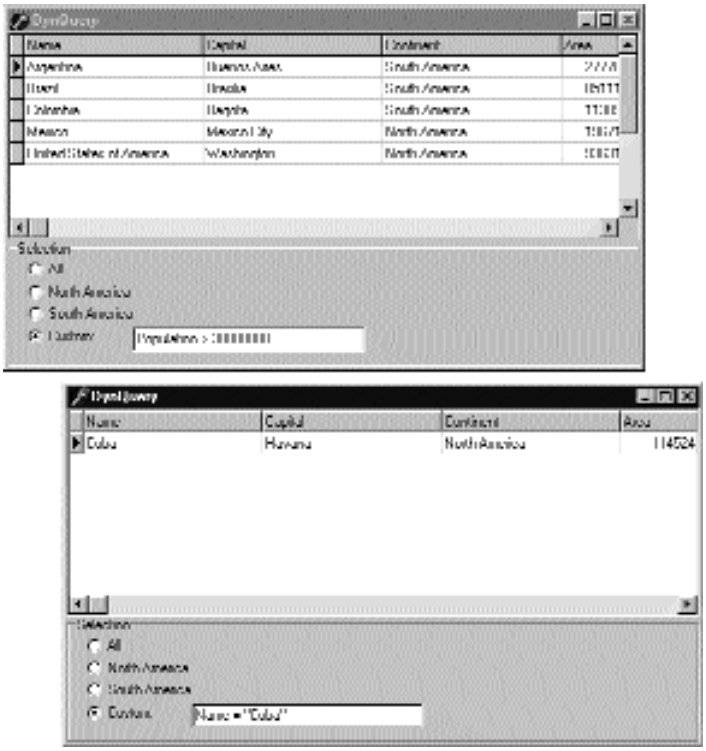
SAVET

U prethodnom SQL iskazu sam koristio znake navoda za stringove. Takođe, možete koristiti apostrofe. Ipak, da biste imali apostrofe unutar Pascal stringa, potrebno je da upotrebite dva uzastopna apostrofa. Zbog toga biste, u prethodnom kodu, imali trostruke pa čak i četvorostruke apostrofe, što bi sigurno dovelo do zabune. ■

Za poslednju opcionu kontrolu potrebno je samo da spojimo unapred određeni iskaz sa tekstom polja za izmene, obrađujući mogući izuzetak:

```
procedure TQueryForm.RadioButton4Click(Sender: TObject);
begin
  Query1.Close;
  if (Editl1Text <> '') then
  begin
    Query1.Sql.Clear;
```

Alternativa upotrebi `where` SQL iskaza je upotreba tabele i određivanje opsega slogova koje želite da uzmete u obzir, ili upotreba svojstva `Filtered`, kao što je pokazano u prethodnom primeru. Flitri su Vam na raspolaganju i za upite, ali prirodan način za filtriranje upita je upotreba SQL iskaza, tako da SQL mehanizam ili SQL server mogu obraditi upit umesto programa. Ovaj metod je naročito pogodan ukoliko se mehanizam baze podataka ili SQL server nalaze na drugom kompjuteru, a ne na kompjuteru sa kojeg dolazi upit, jer će podeliti posao na dve mašine i često će smanjiti mrežni protok, kao što ćemo videti u narednom poglavlju.



SLIKA 10.8 *Dve kopije programa DynQuery, od kojih svaka koristi drugačiju SQL where klauzulu*

Upit sa parametrima

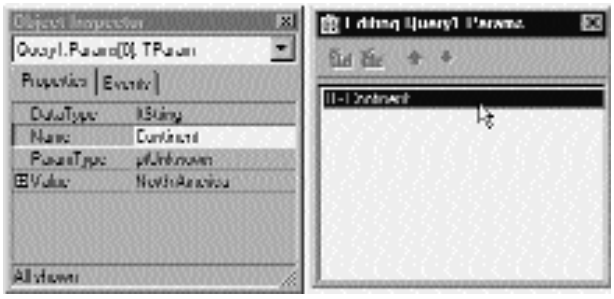
Svi upiti u primeru DynQuery su veoma slični. Umesto da svaki put izrađujemo novi upit, možemo napisati upit sa parametrom i samo promeniti vrednost parametra. Ukoliko odlučimo da odaberemo države kontinenta, na primer, možemo napisati sledeći iskaz:

```
select * from Country where Continet = :Continent
```

U ovoj SQL klauzuli `:Continent` je parametar. Možemo odrediti njegov tip podataka i početnu vrednost koristeći editor `Params` kolekcije svojstava komponente Query. Kada se otvori editor Parameters kolekcije, kao što je prikazano na slici 10.9, videćete listu parametara definisanih SQL iskazom, te odredite tip podataka i početnu vrednost ovih parametara.

Formular koji prikazuje ovaj novi program, program ParQuery, koristi listu umesto opcione kontrole za svaku selekciju. Umesto pripremanja elemenata liste u vreme dizajniranja, izdvajamo sadržaj iz baze podataka kada program počne sa izvršavanjem. Ovo se postiže upotrebom još jedne Query komponente sa SQL iskazom:

```
select distinct Continent from Country
```



SLIKA 10.9 *Editovanje kolekcije parametara Query komponente*

Posle aktiviranja ovog upita program pretražuje skup rezultata, izdvajajući sve vrednosti, i dodaje ih listi:

```
procedure TQueryForm.FormCreate(Sender TObject);
begin
  // get the list of continents
  Query2 Open;
  while not Query2.EOF do
  begin
    ListBox1.Items.Add (Query2.Fields [0].AsString);
    Query2.Next;
  end;
  ListBox1.ItemIndex := 0;

  // open the first query
  Query1.Params[0].Value := ListBox1.Items [0];
  Query1.Open;
end;
```

Pre otvaranja upita program selektuje kao parametar prvi element liste, koji se takođe aktivira određivanjem vrednosti 0 za svojstvo `ItemIndex`. Kada je lista selektovana, program zatvara upit i menja parametar:

```
procedure TQueryForm.ListBoxqClick(Sender: Tobject);
begin
  Query1.Close;
  Query1.Params[0].Value :=
    ListBox1.Items [ListBox1.ItemIndex];
  Query1.Open;
end;
```

Na ovaj način se prikazuju države odabranog kontinenta u listi, kao što možete videti na slici 10.10. Poslednje poboljšanje je to da kada korisnik unese slog sa novim kontinentom, novi

kontinent se automatski dodaje listi. Umesto osvežavanja cele liste, kodom koji se izvršava u metodu `FormCreate`, osvežavanje možemo izvršiti obradom `BeforePost` događaja i dodavanjem kontinenta listi ukoliko se već ne nalazi u listi:

```
procedure TQueryForm.Query1BeforePost(Dataset: TDataSet):
var
  StrhewCont: string;
begin
  // add the continent, if not already in the list
  StrNewCont := Query1.FieldByName ('Continent').Asstring;
  if ListBox1.Items.IndexOf (StrNewCont) < 0 then
    ListBox1.Items.Add (StrNewCont);
end;
```

SLIKA 10.10 *Primer ParQuery u vreme izvršavanja*

Možemo dodati još koda da bismo iskoristili specifične karakteristike parametarskih upita. Ovi upiti se mogu optimizovati da bi brže reagovali na promene parametra, to jest mogu se pripremiti. Jednostavno pozovite metod `Prepare` pre nego što program prvi put otvori upit (posle određivanja vrednosti `False` za svojstvo `Active` komponente Query u vreme dizajniranja) i pozovite `Unprepare` kada nećete više koristiti upit:

```
procedure TQueryForm.FormCreate(Sender: TCbject);
begin
  ...
  // prepare and open the first query
  Query1. Prepare;
  Query1.Params[0] Value := ListBoxlItems [0]
  Query1.Open;
end;

procedure TQueryForm.FormDestroy(Sender: TObject);
begin
  Query1.Close;
  Query1.Unprepare;
end;
```

NAPOMENA

Pripremljeni parametarski upiti su veoma važni kada radite sa velikim tabelama i složenim upititma. Zapravo, da bi optimizovale ovakav upit, mnoge baze podataka kreiraju privremene indekse. Umesto da se indeks kreira svaki put kada se otvara upit, pripremljen upit može podesiti optimizaciju samo jednom na početku, štedeći tako mnogo vremena kada se parametar promeni. ■

Upotreba više tabela

U programima za baze podataka koje smo do sada napisali, koristili smo samo jednu tabelu. U programima koji se svakodnevno koriste, obično se pristupa većem broju tabela. Može postojati glavna tabela sa imenima kupaca i sekundarna tabela sa narudžbinama koje su kupci načinili. Može postojati tabela sa mestima i tabela sa zaposlenima, sa numeričkim ID-om za mesta na kojima zaposleni rade. Postoji neograničen broj primera zavisnosti između tabela baze podataka.

Važna stvar na koju treba obratiti pažnju je ovaj kratak uvod o nekoliko pristupa u Delphiju za povezivanje različitih tabela:

- Master/detail zavisnost između tabela ili upita omogućava Vam da u sekundarnom skupu podataka selektujete samo slogove koji se odnose na aktuelni element master skupa podataka. Na primer, u glavnoj tabeli možete selektovati kupca i prikazati sve porudžbine koje je taj kupac načinio u sekundarnoj tabeli.
- Lookup polje jedne tabele (ili upita) prikazuje vrednost nekog drugog polja u odgovarajućem slogu povezane tabele. Na primer, u tabeli Purchase Orders, ID kupca koji je načinio određenu porudžbinu, može biti lookup polje koje prikazuje ime kupca iz tabele Customers.
- Spajanje naznačeno unutar SQL upita može definisati mnoge druge zavisnosti među tabelama.

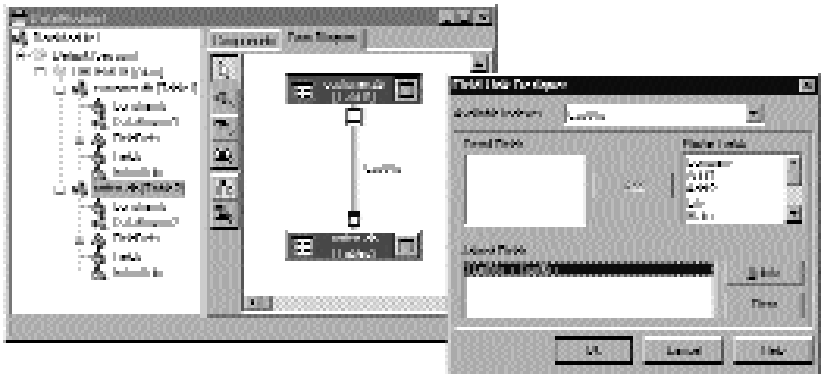
Kako se spajanje u SQL upitu izvršava na serveru, dok se master/detail i lookup zavisnosti izračunavaju na klijentu, SQL upiti imaju više smisla u klijent/server okruženju, a master/detail i lookup veze mogu dati najbolje performanse kada se pristupa lokalnim tabelama. Poslednja dva pristupa umnogome poboljšavaju korisnički interfejs, te se zbog toga mogu koristiti i u klijent/server arhitekturi.

Master/detail sa tabelama

Delphi obezbeđuje nekoliko jednostavnih načina za kreiranje master/detail strukture. Verovatno najjednostavniji način je upotreba Database Form Wizarda i selektovanje master/detail formulara na prvoj strani. (Kao što je istaknuto u Poglavlju 9, Database Form Wizard je toliko jednostavan za korišćenje, da nije potrebno posebno ga objašnjavati u ovoj knjizi.) Gotovo isto toliko je lak i novi pristup koji možemo koristiti u Delphiju 5 da bismo obavili isti zadatak — upotreba pogleda Data Diagram modula podataka. To ćemo uraditi u narednom primeru, u programu MastDet.

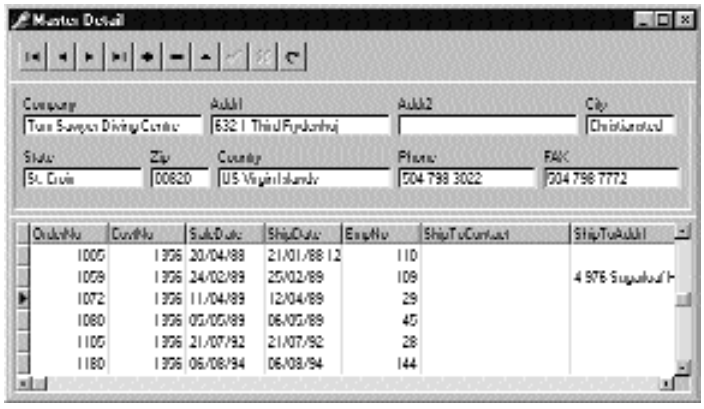
Pošto u Delphiju koristimo primere tabela, nema mnogo različitih mogućnosti primera izrade master/detail formulara. Naš primer će koristiti tabele Customer i Orders, koje takođe koriste i neki Delphi primeri programa. Jednostavno smestite dve Table komponente u modul podataka,

povežite ih sa DBDEMOS alijasom i povežite ih sa dvema tabelama. Sada možete prevući tabele u pogled Diagram, selektovati vezu Master Detail i prevući je od tabele Customer do tabele Orders. Data Module Designer će prikazati okvir za dijalog Field Link Designer, prikazan na slici 10.11, u kome možete definisati kako povezati tabele.



SLIKA 10.11 *Okvir za dijalog Field Link Designer Data Module Designera se aktivira kada izrađujete master/detail zavisnost. Kada definišete takvu zavisnost, možete je modifikovati u bilo kom trenutku ukoliko odaberete Edit u Data Module Designeru.*

Na slici 10.12 možete videti primer glavnog formulara programa MastDet u vreme izvršavanja. Ja sam u gornji deo smestio kontrole koje prepoznaju podatke, a Grid koji je povezan sa tabelom sam smestio u donji deo formulara. Na ovaj način za svaki master slog možete odmah videti listu povezanih detail slogova, u ovom slučaju svih porudžbina koje je načinio klijent. Svaki put kada selektujete novog kupca, tabela ispod će prikazati samo slogove koji sadrže porudžbine koje se odnose na tog kupca.



SLIKA 10.12 *Primer MastDet u vreme izvršavanja*

Kako funkcioniše ovaj program? Odgovor je veoma jednostavan. Ukoliko otvorite modul podataka i pogledate svojstva dveju Table komponenata u Object Inspectoru, videćete sledeća pravila:

```
object Table1: TTable
  DatabaseName = 'DBDEMOS'
  Tablename = 'customer.db'
end
object Table2: TTable
  DatabaseName = 'DBDEMOS'
  TableName = 'orders.db'
  IndexFieldnames = 'CustNo'
  MasterFields = 'CustNo'
  MasterSource = DataSource1
end
```

Druga tabela sadrži master izvor (izvor podataka koji je povezan sa prvom tabelom), i povezuje se sa određenim poljem koje određuje uređenu listu.

Master/detail struktura sa upitima

Prethodni primer je koristio dve tabele za izradu master/detail formulara. Kao alternativu možete definisati ovaj tip spajanja upotrebom SQL iskaza.

Za ovaj primer sam spojio tabelu ORDERS.DB sa tabelom ITEMS.DB koja opisuje svaki element porudžbine. Dve tabele se mogu spojiti upotrebom polja OrderNo. Kada generišete kod, program nazvan Orders se ponaša baš kao i prethodni. Ovoga puta, ipak, trik je u SQL iskazu drugog objekta upita:

```
select
  items."OrderNo",
  items."Itemno",
  items."PartNo",
  items."Qty"
from
  items
where
  "items"."OrderNo" = :"OrderNo"
```

Kao što možete videti, ovaj SQL iskaz koristi parametar `OrderNo`. Ovaj parametrar je direktno povezan sa prvim upitom jer je svojstvo `DataSource` za `Query1` podešeno na `DataSource1`. Drugim rečima, drugi upit se smatra kontrolom podataka koja je povezana sa prvim izvorom podataka. Svaki put kada se promeni slog u prvom izvoru podataka, komponenta `Query2` se ažurira, kao i bilo koja druga komponenta koja je povezana sa `DataSource1`. Polje koje se koristi za vezu, u ovom slučaju, jeste polje koje ima isti naziv kao i parametar upita.

Upotreba Lookup Combo polja

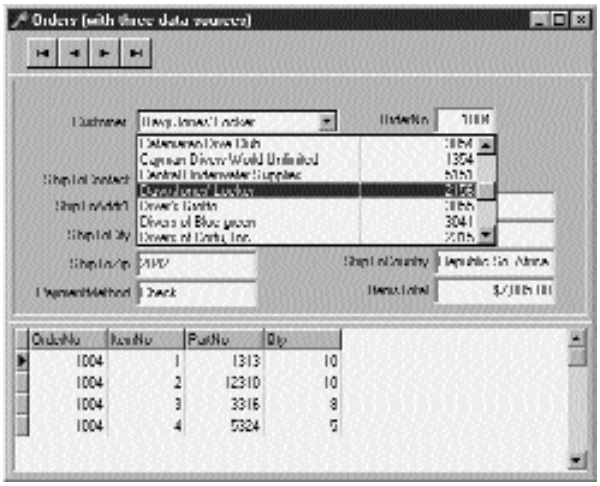
Ukoliko izradite standardni formular za Orders, potrebno je da radite sa brojem kupca porudžbine, što nije najprirodniji način — većina korisnika bi više volela da radi sa imenima kupaca. Ipak, u bazi podataka, imena kupaca se čuvaju u odvojenoj tabeli, da bi se izbeglo dupliranje podataka o kupcima za svaku porudžbinu kupca. Da bih izbegao rad sa brojevima kupaca,

ja sam na formular postavio novu komponentu: kontrolu DBLookupComboBox. Ova komponenta se može povezati sa dva izvora podataka istovremeno: jednim izvorom podataka koji sadrži stvarne podatke i drugim koji podatke prikazuje. Mi želimo da ovu komponentu povežemo sa vrednošću CustNo za `DataSource1`, dakle, sa master upitom, ali da omogućimo da prikazuje informacije izdvojene iz druge tabele, tabele CUSTOMER.DB.

Da bih ovo postigao, ja sam uklonio standardnu DBEdit komponentu povezanu sa brojem kupca i zamenio sam je komponentom DBLookupComboBox i komponentom DBText. DBText je vrsta oznake, ili tekst koji se može menjati. Zatim sam dodao novi izvor podataka (`DataSource3`) povezan sa tabelom (`Table1`) koja se odnosi na fajl CUSTOMER.DB. Da bi program funkcionisao, potrebno je da odredite nekoliko svojstava komponente `DBLookupComboBox1`. Evo liste relevantnih vrednosti:

```
object DBLookupComboBox1: TDBLookupComboBox
  DataField = 'CustNo'
  DataSource = DataSource1
  KeyField = 'CustNo'
  ListField = 'Company'
  ListSource = DataSource3
end
```

Prva dva svojstva određuju glavnu vezu, kao i obično. Preostala tri svojstva određuju sekundarni izvor (`ListSource`), polje koje se koristi za spajanje (`KeyField`) i informaciju koja se prikazuje (`ListField`). Pored unošenja naziva jednog polja možete uneti i nazive više polja. Kao tekst combo polja se prikazuje samo prvo polje, ali ukoliko odredite veliku vrednost za svojstvo `DropDownWidth`, lista combo polja će uključiti više kolona podataka. Na slici 10.13 možete videti izlaz programa.



SLIKA 10.13 *Izlaz primera Orders kada DBLookupComboBox prikazuje više polja u listi*

SAVET

Ukoliko odredite indeks tabele povezane sa DBLookupComboBoxom na polje Company, lista će prikazivati kompanije po abecednom redu umesto brojeva kupaca. To je ono što sam ja učinio u primeru. ■

Šta je sa kodom ovog programa? Pa, nema ga. Sve funkcioniše kada podesite odgovarajuća svojstva. Tri spojena izvora podataka ne zahtevaju kod. Ovo pokazuje da se upotreba master/detail i lookup veza može brzo podesiti i da je veoma efikasna. Jedini stvarni nedostatak je taj da se ove tehnike, naročito lookup, ne mogu koristiti kada broj slogova postane preveliki, naročito u mrežnom i klijent/server okruženju. Premeštanje stotina slogova, samo da bi se dobilo lepo lookup combo polje, nije naročito efikasno.

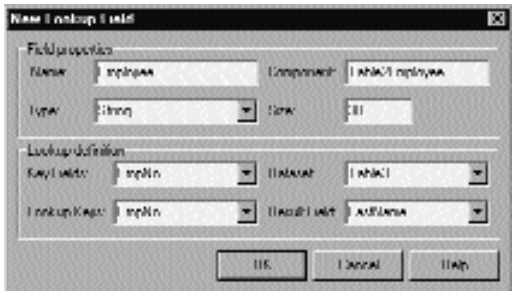
Lookup u tabeli

Umesto smeštanja DBLookupComboBoxa na formular, možemo smestiti lookup listu u komponentu DBGrid. Da bismo dodali fiksirane selekcije u DBGrid, možemo jednostavno editovati `PickList`, podsvojstvo svojstva `Columns`. Da bismo prilagodili tabelu sa živim lookupom, potrebno je definišemo lookup polje upotrebom Fields editora.

Kao primer, ja sam uzeo program MastDet i pretvorio ga u MastDet2. U prvobitnom programu tabela je prikazivala broj zaposlenog koji je preuzeo porudžbinu. Zašto ne bismo umesto toga prikazali ime zaposlenog i omogućili korisniku da ga odabere iz liste zaposlenih?

Da bih ovo postigao, ja sam dodao modul podataka dvema komponentama: Table i DataSource, koje se odnose na `EMPLOYEE.DB` tabelu baze podataka. Zatim sam otvorio Fields editor za tabelu `ORDERS` i dodao sam sva polja. Selektovao sam polje `EmpNo` i odredio vrednost `False` za svojstvo `Visible` da bih ga uklonio iz tabele (ne možemo ga sasvim ukloniti jer se koristi za povezivanje sa odgovarajućim poljem tabele Employee).

Sada je vreme da definišemo lookup polje. Ukoliko ste pratili prethodne korake, možete preći na karticu Data Diagram i prevući lookup zavisnost iz tabele `ORDERS` u tabelu `EMPLOYEE`, povezujući ih u rezultujućem okviru za dijalog (videti sliku 10.14). Sličan okvir za dijalog možete aktivirati upotrebom komande New Field editora Fields.

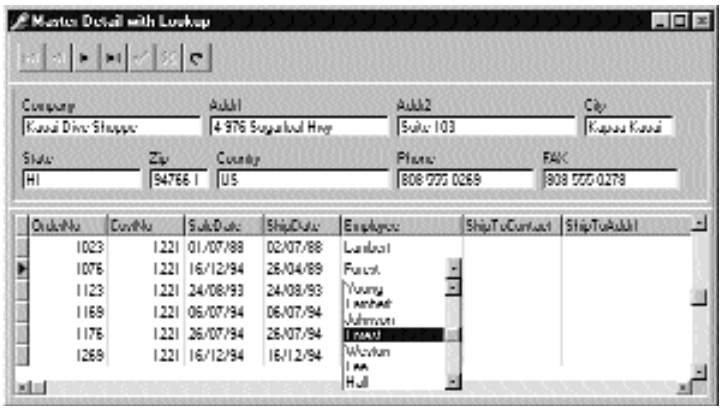


SLIKA 10.14 *Okvir za dijalog New Lookup Field je aktiviran prevlačenjem lookup veze između dva skupa podataka Data Diagrama*

Vrednosti koje određujete u okviru za dijalog New Lookup Field uticaće na svojstva novog `TField` dodatog tabeli, što je pokazano DFM opisom polja koje odgovara vrednostima prikazanim na slici 10.14:

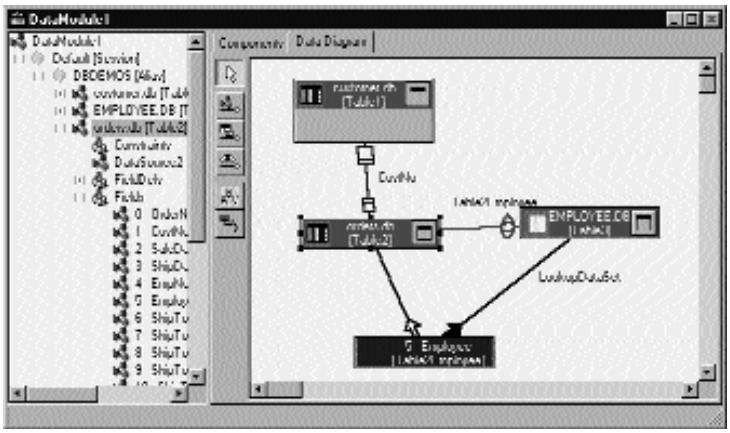
```
object Table2Employee: TStringField
  FieldKind = fkLookup
  FieldName = 'Employee'
  LookupDataSet = Table3
  LookupKeyFields = 'EmpNo'
  LookupResultField = 'LastName'
  KeyFields = 'EmpNo'
  FixedChar = False
  Size = 30
  Lookup = True
end
```

To je sve što je potrebno da bi lista funkcionisala (videti sliku 10.15) i da bi se prikazala vrednost polja u vreme dizajniranja. Primetićete da nije potrebno prilagoditi svojstvo `Columns` tabele jer se kontrola i vrednost sedam redova uzimaju po definiciji. To ne znači da ne možete koristiti ovo svojstvo za dalje prilagođavanje ovih i drugih vizuelnih elemenata tabele.



SLIKA 10.15 *Izlaz primera MastDet2, sa listom unutar tabele koja prikazuje vrednosti koje se dobijaju iz druge tabele baze podataka*

Nova master/detail zavisnost će biti jasno vidljiva u pogledu Data Diagram Data Module Designera. Ukoliko ovom pogledu dodate lookup polje, njegov izgled će biti detaljniji, kao što možete videti na slici 10.16.



SLIKA 10.16 *Master/detail zavisnost u pogledu Data Diagram. Zapazite veze lookup polja.*

Napredna upotreba kontrole DBGrid

U mnogim primerima ovog i prethodnog poglavlja koristili smo kontrolu DBGrid, jer jednostavno obezbeđuje zgodan način prikazivanja informacija o više polja i slogova istovremeno. Za razliku od većine drugih kontrola koje prepoznaju podatke, koje su jednostavne za upotrebu, DBGrid ima mnoge opcije i mnogo je moćnija nego što zamišljate.

Nekoliko narednih odeljaka se odnosi na naprednije operacije koje možete obaviti upotrebom kontrole DBGrid. Prvi primer pokazuje kako da nacrtate tabelu, drugi kako da klonirate ponašanje polja za potvrdu za Boolean selekciju unutar tabele, a poslednji primer pokazuje kako da upotrebite mogućnost višestrukog selektovanja u tabeli.

Iscrtavanje DBGrida

Postoji mnogo razloga za prilagođavanje izgleda tabele. Dobar primer je isticanje određenih polja ili slogova. Drugi primer je određivanje izgleda polja koja se obično ne prikazuju u tabeli, kao što su BLOB, grafika ili memo polja.

Da biste detaljno prilagodili iscrtavanje kontrole DBGrid, potrebno je da odredite vrednost `False` za svojstvo `DefaultDrawing` i da obradite događaj `OnDrawColumnCell`. Zapravo, ukoliko ostavite vrednost `True` za `DefaultDrawing`, tabela će se prikazati kako je to unapred određeno, izuzev ukoliko odlučite da je iznova iscrtate, čime će se utrošiti dodatno vreme i videće se treptanje.

Alternativni pristup je pozivanje metoda `DefaultDrawColumnCell`, posle promene fonta ili ograničavanja izlaznog četvorougla. U poslednjem slučaju možete obezbediti dodatno iscrtavanje u ćeliji i prepustiti tabeli da ispuni preostalu oblast standardnim izlazom. To je ono što sam ja učinio u programu DrawData.

DBGRid kontrola u ovom primeru, koja je povezana sa `BIOLIFE` tabelom baze podataka DBDEMOS, sadrži sledeća svojstva:

```
object DBGrid1: TDBGrid
  Align = alClient
  DataSource = DataSource1
  DefaultDrawing = False
  Font.Height = -16
  Font.Name = 'MS Sans Serif'
  Font.Style = [fsBold]
  TitleFont.Height = -11
  TitleFont.Name = 'MS Sans Serif'
  Title.Font.Style = []
  OnDrawColumnCell = DBGrid1DrawColumnCell
end
```

Obrada događaja `OnDrawColumnCell` se poziva jednom za svaku ćeliju tabele i ima nekoliko parametara, uključujući i četvorougao koji odgovara ćeliji, indeks kolone koju treba da iscrtamo, samu kolonu (sa poljem, poravnanjem i drugim podsvojstvima) i status ćelije. Kako da boja određenih ćelija bude crvena? Možemo je promeniti u specijalnim slučajevima:

```
procedure TForm1.DBGrid1DrawColumnCell(Sender: TObject;
  const Rect: TRect; DataCol: Integer; Column: TColumn;
  State: TGridDrawState);
begin
  // red font color if length > 100
  if (Column.Field = Table1Lengthcm) and
      (Table1Lengthcm.AsInteger > 100) then
    DBGrid1.Canvas.Font.Color := clRed;

  // default drawing
  DBGrid1.DefaultDrawDataCell (Rect, Coluinn.Field, State);
end;
```

Sledeći korak je iscrtavanje memo i grafičkih polja. Za memo jednostavno možemo implementirati memo `OnGetText` i `OnSetText` događaje. Zapravo, tabela će čak dozvoliti editovanje memo polja ukoliko događaj `OnSetText` nije `nil`. Evo koda dveju obrada događaja. Ja sam koristio `Trim` da bih uklonio karaktrere koji se ne štampaju, što doprinosi da tekst bude prazan prilikom editovanja:

```
procedure TForm1.Table1NotesGetText(Sender: TField;
  var Text: String; DisplayText: Boolean);
begin
  Text := Trim (Sender.AsString);
end;

procedure TForm1.Table1NotesSetText(Sender: TField;
  const Text: String);
begin
  Sender.AsString := Text;
end;
```

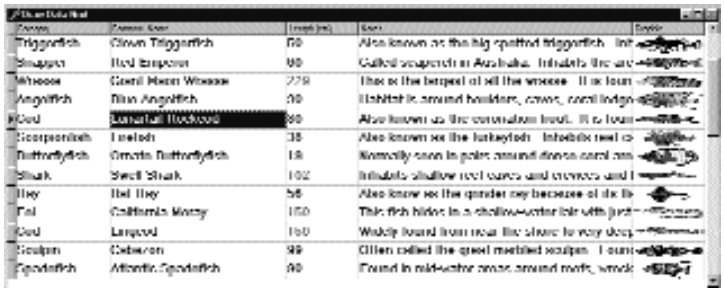
Za sliku je najjednostavnije kreirati privremeni `TBitmap` objekat, dodeliti mu grafičko polje i iscrtati bitmapu u `Canvas` tabele. Kao alternativu sam uklonio grafičko polje iz tabele, određujući vrednost `False` za svojstvo `Visible`, i dodao sam sliku nazivu ribe sledećim kodom u obradi `OnDrawColumnCell` događaja:

```
var
  Bmp: TBitmap;
  OutRect: TRect;
  BmpWidth: Integer;
begin
  // default output rectangle
  OutRect := Rect;

  if Column.Field = Table1Common_Name then
  begin
    // draw the image
    Bmp := TBitmap.Create;
    try
      Bmp.Assign (Table1Graphic);
      BmpWidth := (RectBottom - Rect.Top) * 2;
      OutRect.Right := Rect.Left + BmpWidth;
      DBGrid1.Canvas.StretchDraw (OutRect, Bmp);
    finally
      Bmp.Free;
    end;
    // reset output rectangle, leaving space for the graphic
    OutRect := Rect;
    OutRectLeft := OutRect.Left + Bmpwidth;
  end

  // red font color if length > 100 (omitted — see above)

  // default drawing
  DBGrid1.DefaultDrawDataCell (OutRect, Column.Field, State);
```

Kao što možete videti iz prethodnog koda, program prikazuje sliku u malom četvorouglu na levoj strani tabele, a zatim menja izlazni četvorougao u ostatak površine pre nego što se aktivira iscrtavanje. Efekat možete videti na slici 10.17.

NAPOMENA

U primeru sam koristio veliki font da bih povećao visinu redova. Bilo bi lepo da imamo mogućnost podešavanja visine redova, ali to nije jednostavno. "Delphi Developer's Handbook" (Sybex, 1998) opisuje razvoj proširene komponente DBGrid sa promenljivom visinom redova i daje listing izvornog koda. Sa mog web sajta možete preuzeti kompajliranu verziju komponente. ■



SLIKA 10.17 *Program DrawData prikazuje tabelu koja sadrži tekst memo polja i Borlandove ribe*

Ćelija sa poljem za potvrdu

Još jedno uobičajeno proširenje kontrole DBGrid, koje se može videti u mnogim komponentama nezavisnih programera, jeste upotreba polja za potvrdu za određivanje statusa Boolean polja. Da biste ovo jednostavno postigli, postavite kontrolu DBCheckBox ispred tabele kada korisnik odabere odgovarajući element. Ja sam to učinio u proširenju primera FldText iz poslednjeg poglavlja.

Formular koji prikazuje novi program, nazvan CheckDbg, sadrži samo tabelu i polje za potvrdu, a zasnovan je na tabeli baze podataka koju možete ispuniti upotrebom primera DBAware iz poslednjeg poglavlja. Ovo je rezime tekstualnog opisa formulara:

```
object DbaForm: TDbaForm
OnCreate = FormCreate
  object DBGrid1: TDBGrid
    Align = alClient
    DataSource = DataSource1
    OnColEnter = DBGridlColEnter
    OnDrawColumnCell = DBGrid1DrawColumnCell
    OnKeyPress = DBGridlKeyPress
  end
  object DBCheckBox1: TDBCheckBox
    Caption = 'Senior'
    DataField = 'Senior'
    DataSource = DataSource1
    ValueChecked = 'True'
    ValueUnchecked = 'False'
    Visible = False
  end
  object Table1: TTab1e
    DatabaseName = 'DBDEMOS'
    TableName = 'Workers'
  end
end
```

Primetićete da je polje za potvrdu inicijalno sakriveno i da program obrađuje nekoliko događaja kontrole DBGrid. Prvi je događaj `OnDrawColumnCell`, koji se ne koristi za prilagođavanje prikaza (određena je vrednost `True` za svojstvo `DefaultDraving`), ali samo za izračunavanje pozicije polja za potvrdu kada je ćelija odgovarajućeg polja selektovana:

```
procedure TDbaForm.DBGrid1DrawColumnCell(Sender TObject;
  const Rect: TRect; DataCol: Integer; Column: TColumn;
  State: TGridDrawState);
begin
  if (gdFocused in State) and
    (Column.Field = Table1Senior) then
  begin
    DsCheckBox1.SetBounda (
    Rect.Left + DBGrid1.Left + 1,
    Rect.Top + DBGrid1.Top + 1,
    Rect.Right - Rect.Left,
    Rect.Bottom - Rect.Top);
  end;
end;
```

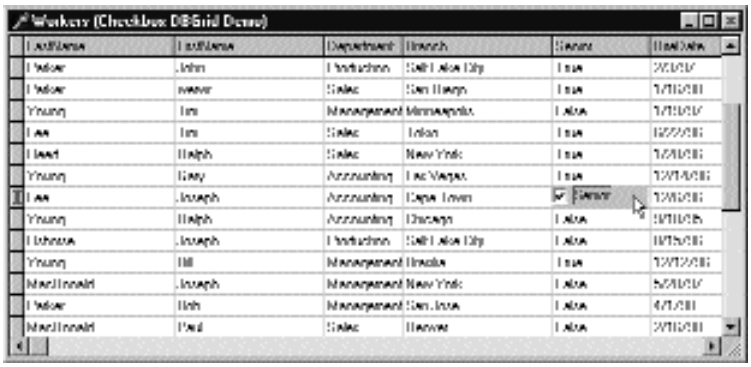
Polje za potvrdu je prikazano ili sakriveno kada korisnik pristupa odgovarajućoj koloni ili izlazi iz nje, što se odigrava u događaju OnColEnter. Primetićete da se na kolonu ne možemo referisati po poziciji jer korisnik može da premesti kolone:

```
procedure TDbaForm.DBGrid1ColEnter(Sender: TObject);
begin
  if DBGrid1.Columns [DBGrid1.SelectedIndex].
      Field = Table1Senior then
    DBCheckBox1.Visible := True
  else
    DBCheckBox1.Visible := False;
end;
```

Konačno, kao dodatno poboljšanje, kada je polje za potvrdu vidljivo (to jest, kada je korisnik aktivirao odgovarajuće polje), program presreće unos sa tastature u tabelu, menjajući stanje polja za potvrdu umesto da prihvati unos:

```
procedure TDbaForm.DBGrid1ColEnter(Sender: TObject);var: Char)
begin
  if DBCheckBox1.Visible and (Ord (Key) > 31) then
  begin
    Key := #0;
    Table1.Edit;
    DBCheckBox1.Checked := not DBCheckedBox1.Checked;
      DBCheckBox1.Field.AsBoolean := DBCheckedBox1.Checked;
  end;

end;
```

Da bi ovo funkcionisalo, ne samo da moramo da menjamo status polja za potvrdu, već, takođe, moramo da pređemo u mod edit i moramo da ažuriramo podatke polja. Primer izlaza programa možete videti na slici 10.18.

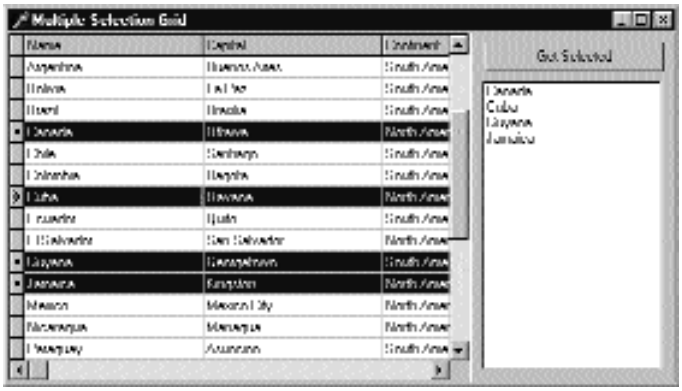


SLIKA 10.18 *Tabela primera CheckDbg koristi polje za potvrdu za selektovanje vrednosti Boolean polja*

Tabela koja dozvoljava višestruko selektovanje

Treći i poslednji primer prilagođavanja kontrole DBGrid odnosi se na višestruko selektovanje. DBGrid možete podesiti tako da korisnik može selektovati više redova (to jest, više slogova). Ovo je veoma lako jer sve što je potrebno da učinite, jeste da promenite element `dgMultiSelect` svojstva `Options` tabele. Kada ste selektovali ovu opciju, korisnik može držati pritisnut taster Ctrl i mišem označiti više redova tabele, a efekat možete videti na slici 10.19.

Pošto tabela baze podataka može imati samo jedan slog aktivan, koja informacija se čuva u tabeli za selektovane elemente? Tabela jednostavno čuva listu oznaka za selektovane slogove. Ova lista, koja je tipa `TBookmarkList`, dostupna je preko svojstva `SelectedRows`. Pored pristupanja brojnim objektima liste preko svojstva `Count`, svaku oznaku možete dobiti preko svojstva `Items`. Svaki element liste je tipa `TBookmarkList` koji predstavlja pokazivač oznake koji možete da dodelite `Bookmark` svojstvu tabele.



SLIKA 10.19 *Primer MltGrid sadrži kontrolu DBGrid koja dozvoljava selekciju više redova*

NAPOMENA

`BookmarkStr` je string tip, ali podatke treba posmatrati kao proširive. Ne bi trebalo da se oslonite na bilo koju strukturu podataka koju možete pronaći ukoliko pogledate vrednost oznake, i ne bi trebalo da predugo zadržavate podatke, ili ih čuvajte u zasebnom fajlu. Podaci oznake mogu varirati u zavisnosti od drajvera baze podataka i konfiguracije indeksa, a mogu postati neupotrebljivi kada se dodaju ili uklone slogovi iz skupa podataka (kada to Vi ili neko drugi učini). ■

Da bismo rezimirali korake, evo koda primera MltGrid, koji se aktivira kada kliknete kontrolu za pomeranje Name polja selektovanih slogova liste:

```
procedure TForm1.Button1Click (Sender: TObject);
var
  I: Integer;
  BookmarkList: TBookmarkList;
  Bookmark: TBookmarkStr;
begin
  // store the current position
  Bookmark := Table1.Bookmark;
```

```
  try
    // empty the list box
    ListBox1.Items. Clear;
    // get the selected rows of the grid
    BookmarkList := DbGrid1.SelectedRows
    for I := 0 to BookmarkList.Count — 1 do
  begin
    // for each, move the table to that record
    Table1.Bookmark := BookmarkList[I];
    // add the name field to the listbox
    ListBox1.Items.Add (Table1.FieldByName (
      'Name' ).Asstring);
    end;
  finally
    // go back to the initial record
    Table1.Bookmark := Bookmark;
  end;
end;
```

Data Dictionary

Veoma je česta upotreba polja sa sličnim izgledom (na primer, sa istom maskom za prikazivanje) u jednoj ili više aplikacija. Ukoliko koristite celobrojne brojeve, decimalne brojeve, procente, telefonske ili faks brojeve (verovatno isti broj sa različitim ekstenzijama) i druga standardna polja, veoma je monotono podešavati ih jedno po jedno. Zbog toga Delphi sadrži Data Dictionary. To je vrsta baze podataka koja čuva svojstva polja. Svojstva ovih standardnih polja možete definisati koristeći Dictionary, ili ih možete jednostavno kopirati iz postojećih polja (naravno, takođe možete koristiti postojeće elemente Data Dictionary).

U klijent/server okruženju (kada nekoliko Delphi programera radi na projektu), Data Dictionary se može nalaziti na više servera za dodatno deljenje informacija.

NAPOMENA

Unapred odreden Data Dictionary je implementiran preko Paradox tabele, ali možete definisati novi na osnovu SQL server tabele. ■

Data Dictionary i Fields editor

Većina operacija koje koriste Data Dictionary se obavlja u Fields editoru tabele ili upita. Lokalni meni Fields editora sadrži pet komandi koje se odnose na upotrebu baze Data Dictionary (videti sliku 10.20).

SLIKA 10.20 *Lokalni meni Fields editora sa komandama menija koje se koriste za komunikaciju sa bazom Data Dictionary*

Operacija	Značenje
Associate Attributes	Ova komanda se koristi za pridruživanje skupa atributa za dato polje. U praksi možete selektovati jedan od skupova atributa iz baze Dictionary koji će se koristiti za aktuelno polje. Kada polje udružite sa skupom atributa iz baze Data Dictionary, atributi će biti kopirani u odgovarajuća svojstva polja.
Unassociate Attributes	Ova komanda je suprotna komandi Associate Attributes. Uklanja udruživanje između polja i skupa atributa.
Retrieve Attributes	Ova komanda se koristi za dobijanje trenutnih vrednosti skupa atributa. Može se koristiti samo kada je polju pridružen skup atributa. Ovu komandu možete posmatrati kao komandu za učitavanje.
Save Attributes	Ova komanda je suprotna komandi Retrieve Attributes. Komanda kopira vrednosti svojstava aktuelnog polja u odgovarajući skup atributa. Ukoliko nije odabran nijedan skup atributa, biće zatraženo da unesete naziv novog skupa, kao što je to slučaj sa komandom Save Attributes As.
Save Attributes As	Ova komanda se koristi za pridruživanje polja novom skupu atributa, za koji treba da unesete naziv u okvir za dijalog koji će se prikazati.

Ove komande bi trebalo da budu intuitivne, pa predlažem da radite sa Data Dictionary da biste naučili kako da ih koristite. Veoma brzo ćete se navići na Data Dictionary. Definisanje veza između tipa atributa i jednog ili više polja tabela primorava Delphi da koristi odgovarajuće atribute svaki put kada koristite tabelu sa nekim od tih polja. Na primer, sva Phone polja tabele (kao što je broj telefona ili broj faksa) mogu se pridružiti određenom skupu atributa, tako da svaki put kada koristite tabelu u programu, Delphi automatski dodeli odgovarajuću ulaznu masku za polje, kao i ostale atribute. Ovo se dešava i kada se polje dobija iz upita, ali samo ukoliko kreirate polje u vreme dizajniranja.

Šta je skup atributa?

U prethodnom odeljku koristio sam termin skup *atributa* nekoliko puta. Skup atributa je element (slog) iz Data Dictionary. Skup atributa se odnosi na nekoliko svojstava `TField` objekta, ali, takođe, sadrži i druga opšta svojstva. Mnoga svojstva skupa atributa se odnose na različite `TField` potklase; ovo bi trebalo lako razumeti. Evo abecedne liste ovih svojstava:

```
Alignment           EditMask
BlobType            MaxValue
Currency            MinValue
DisplayFormat       Precision
DisplayLabel        ReadOnly
DisplayValues       Required
DisplayWidth        Transliterate
EditFormat          Visible
```

Naravno, ukoliko svojstvo odredite kao `Precision` (vrednost se koristi samo za brojeve u pokretnom zarezu), a zatim pridružite skup atributa `TStringField`, vrednost će biti ignorisana.

Nekoliko drugih vrednosti skupa atributa definiše opšte ponašanje polja i koristi se za određivanje načina kreiranja novog objekta polja za dato polje tabele ili upita:

Atribut	Upotreba
`TField` klasa	Označava tip polja (TField potklasa) koji treba kreirati kada se polje dodaje skupu podataka.
`TControl` klasa	Određuje tip komponente koja prepoznaje podatke, a koju će Delphi kreirati kada prevučete polje iz Fields editora na formular. Ukoliko nije data nikakva vrednost, Delphi će koristiti standardni pristup, koji zavisi od tipa polja.
Based On	Ovo je vrednost koju treba da obezbedite kada skup podataka čuvate pod novim imenom. Označava skup atributa na osnovu kojeg se zasniva trenutni skup podataka. To znači da ukoliko načinite izmenu atributa prvobitnog skupa, promena će se odraziti i na aktuelni skup. Analogija koja Vam može pasti na pamet je analogija sa programima za obradu teksta u kojima se koriste stilovi paragrafa. Naravno, ovo podseća i na nasleđivanje.

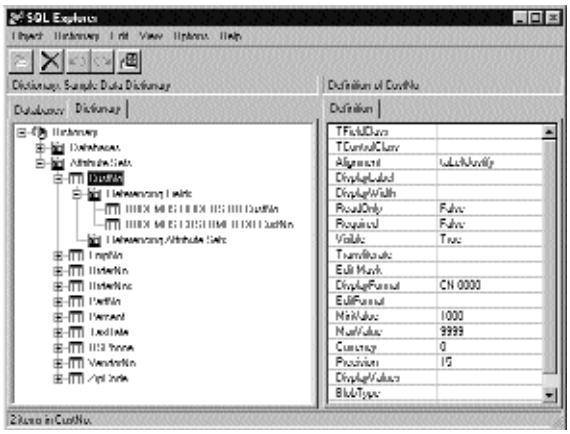
Pretraživanje baze Data Dictionary

Lako možete definisati novi skup atributa tako što ćete sačuvati atribute aktuelnog polja i Fields editora, kao što sam to ranije istakao. Ipak, novi skup atributa možete definisati ili ih jednostavno prikazati iz SQL Explorera (nazvanog Database Explorer u Professional verziji Delphija). Otvorite ovaj alat, odaberite karticu Dictionary iznad levog panela i videćete Default Data Dictionary (DefaultDD), koji je zasnovan na BDESDD Paradox tabeli. Ukoliko ste dodali nove baze Data Dictionary, sve će biti prikazane. Pod stavkom Dictionary pronaći ćete dva poddrveta, Databases i Attribute Sets, kao što je prikazano na slici 10.21.

U poddrvetu Attribute Sets ćete pronaći listu skupova i za svaki od njih vrednosti njegovih svojstava. Takođe postoje i liste tabela baze podataka koje koriste svaki od skupova, i liste drugih

skupova atributa koji su zasnovani na tim skupovima. Takođe, možete koristiti SQL Explorer za kreiranje novih skupova atributa ili za izmenu postojećih.

Veze između polja tabela baze podataka i skupova atributa može se prikazati i izmeniti preko drugog poddrveta, Databases. Kada selektujete polje tabele, combo polje će Vam omogućiti da polju pridružite jedan od mogućih skupova atributa iz Data Dictionary.



SLIKA 10.21 *Data Dictionary prikazan u SQL Exploreru*

Kada pokrećete novi projekat i kada ste planirali tabele baze podataka, ja Vam predlažem da podesite neke skupove atributa i njihove asocijacije pre nego što započnete rad u Delphiju. Ukoliko Vaš plan nije dobro definisan, trebalo bi da upotrebite Fields editor za izradu Data Dictionary uz Vaše tabele, a da zatim upotrebite Explorer za reviziju trenutne situacije i dokumentovanje Vaših izmena.

Obrada grešaka baze podataka

Još jedan važan element programiranja baza podataka jeste obrada grešaka baze podataka. Naravno, možete prepustiti Delphiju da prikaže poruku o izuzetku svaki put kada se desi greška, ali možete pokušati da korigujete greške ili da jednostavno prikažete više detalja o grešci. U osnovi, postoje tri pristupa koja možete upotrebiti za greške baze podataka:

- Možete rizične operacije smestiti u `try-except` blok, operacije kao što su poziv metoda `Open` komponente Query ili metoda `Post` skupa podataka. Ovo nije moguće kada operaciju generiše kontrola koja prepoznaje podatke.
- Možete instalirati obradu `OnExcept` događaja globalnog objekta `Application` ili upotrebiti komponentu ApplicationEvents, što će biti opisano u narednom primeru.
- Možete obraditi određene događaje skupa podataka koji se odnose na greške, kao što su `OnPostError`, `OnEditError`, `OnDeleteError` i `OnUpdateError`. Ove događaje ćemo razmatrati kasnije.

Dok većina klasa izuzetaka u Delphiju jednostavno prikazuje poruku o grešci, kada su u pitanju greške baze podataka, dobićete listu grešaka koja prikazuje lokalne BDE kodove greškaka, a takođe i kodove grešaka SQL servera sa kojim ste povezani. Pored svojstva `Message`, klasa `DBEngineError` sadrži još dva svojstva, `ErrorCount` i `Errors`. Poslednje svojstvo je spisak grešaka:

```
poperty Errors [Index: Integer]: TDBError;
```

Svaki element ove liste je objekat klase `TDBError`, koji ima sledeća svojstva:

```
type
  TDBError = class
  ...
public
  property Category: Byte read GetCategory;
  property ErrorCode: DBIResult read FErrorCode;
  property SubCode: Byte read GetSubCode;
  property Message: string read FMessage;
  property NativeError: Longint read FNativeError;
end;
```

Ja sam koristio ovu informaciju pri izradi jednostavnog programa za bazu podataka koja prikazuje detalje grešaka za memo komponentu. Da bi obradio sve greške, primer DBError instalira obradu `OnException` događaja komponente ApplicationEvents. Obrada događaja jednostavno poziva određeni metod koji se koristi za prikazivanje detalja greške baze podataka, u ovom slučaju `EDBEngineError`:

```
procedure TForm1.ApplicationEvents1Exception (Sender: TObject; E:
Exception);
begin
  Beep;
  if E is EDBEngineError then
    ShowError (EDBEngineError (E))
  else
    ShowMessage (E.Message);
end;
```

Ja sam odlučio da odvojim kod koji se koristi za prikazivanje greške da bih Vam olakšao kopiranje koda i njegovu upotrebu u različitim kontekstima. Evo koda metoda `ShowError`, koji prikazuje sve dostupne informacije komponente `Memo1` koju sam dodao formularu:

```
procedure TForm1.ShowError(E: EDBEngineError);
var
  I: Integer;
begin
  Memo1.Lines.Add('');
  Memo1.Lines.Add('Error: '+ (E.Message));
  Memo1.Lines.Add('Number of errors: ' +
    IntToStr(E.ErrorCount));
  // iterate through the Errors records
  for I := 0 to E.ErrorCount - 1 do
  begin
    Memo1.Lines.Add('Message: ' +
      E.Errors[I] Message);
    Memo1.Lines.Add(' Category: ' +
      IntToStr(E.Errors[I].Category));
```

```
      Memo1.Lines.Add('Error Code: ' +
        IntToStr(E.Errors [I].ErrorCode));
      Memo1.Lines.Add('SubCode: ' +
        IntToStr(E.Errors[I].SubCode));
      Memo1.Lines.Add('Native Error: ' +
        IntToStr(E.Errors[I].NativeError));
      Memo1.Lines.Add('');
    end;
  end;
```

Pored ovog koda za obradu greške, program sadrži tabelu i upit, kao i odgovarajuće obrade događaja koje se odnose na greške. Kao što sam pomenuo, Vi možete instalirati obradu događaja koja se odnosi na specifične greške skupa podataka. Događaji `OnPostError`, `OnDeleteError` i `OnEditError` imaju istu strukturu. Njihove obrade kao parametar dobijaju skup podataka, samu grešku, i akciju koju zahtevate od sistema; to može biti `daFail`, `daAbort` ili `daRetry`:

```
procedure TForm1.Table1PostError(DataSet: TDataSet; E: EDatabaseError;
  var Action: TDataAction);
begin
  Memo1.Lines.Add (' -> Post Error: ' + E.Message);
end;
```

Ukoliko ne navedete akciju, kao u prethodnom kodu, koristi se akcija `daFail`, a izuzetak se prosleđuje globalnoj obradi. Upotreba `daAbort` zaustavlja izuzetak i može se koristiti ukoliko Vaša obrada već prikazuje grešku.

Konačno, ukoliko imate način da odredite uzrok greške i ispravite je, možete upotrebiti akciju `daRetry`.

NAPOMENA

Četvrti događaj greške, `OnUpdateError`, ima drugačiju strukturu i koristi se sa keširanim ažuriranjem jer se informacija šalje nazad od lokalnog keša do baze podataka. Ova obrada je važna prilikom obrade konflikata ažuriranja između različitih korisnika, kao što ćemo to pokazati u narednom primeru. ■

Primer, takođe, sadrži DBGrid povezan sa tabelom. DBGrid možete upotrebiti da biste izvršili neke nedozvoljene opracije, kao što su dodavanje novog sloga sa istim ključem kao postojeći slog ili pokušaj da izvršite nedozvoljeni SQL upit. Kada kliknete neku od četiri kontrole na levoj strani memo polja, generisaćete grešku, kao što se može videti na slici 10.22.

SLIKA 10.22 *Treća kontrola DBError formulara generiše izuzetak sa 17 grešaka baze podataka*

Višekorisničke Paradox aplikacije

Do sada smo videli aplikacije koje se izvršavaju na jednom kompjuteru. U Poglavlju 11 ćemo videti kako da koristimo SQL servere, koji Vam omogućavaju da kreirate aplikacije za veliki broj korisnika. Međurešenje je, kada imate ograničen broj korisnika (obično ne više od dvanaest) koji istovremeno rade sa skupom podataka, upotreba lokalnih Paradox ili Access fajlova koji su deljeni na mreži.

U završnom delu ovog poglavlja ja ću pokazati nekoliko tehnika koje možete upotrebiti kada delite Paradox podatke u višekorisničkom okruženju. Kao deo razmatranja obradiću nekoliko pridruženih tehnika, kao što su pakovanje tabela, BDE callback funkcija, oporavak od pada i konkurentnost.

BDE niskog nivoa

Pre nego što nastavimo, potrebno je da se osvrnemo na ulogu BDE-a i na neke od njegovih karakteristika niskog nivoa koje nisu dostupne preko Delphi komponenata. Kao što sam pomenuo u prethodnom poglavlju, Borland Database Engine je sredstvo za pristupanje podacima baze podataka iz Delphija (izuzev ukoliko ne upotrebljavate korisnički skup podataka). Ovaj mehanizam prima zahtev iz Vaših Delphi programa i, korišćenjem odgovarajućeg drajvera, prevodi zahteve u komande koje prepoznaje baza podataka koju koristite.

BDE je zapravo skup DLL-ova (koji se moraju instalirati uz aplikaciju) koji zapravo daju programerima pristup API-ju niskog nivoa, koji je poznatiji kao IDAPI (Independent Database Application Programming Interface). Naravno, Delphi programeri obično ne koriste pozive ovih API funkcija (baš kao što obično ne pozivaju direktno Windows API), već koriste komponente koje sadrže većinu uobičajenih poziva, kao što su komponente `TTable`, `TQuery`, `TDatabase` i `TSession`. Samo komponente koje prepoznaju podatke čine deo VCL-a i one se ne odnose direktno na BDE.

Ponekad će biti potrebno da koristite neke od BDE karakteristika niskog nivoa koje nisu dostupne preko Delphi komponenata, baš kao što je to slučaj sa Windows API funkcijama. Mi ćemo koristiti nekoliko BDE funkcija niskog nivoa u poslednjem delu ovog poglavlja. Da bismo bili u mogućnosti da ih koristimo, potrebno je da shvatimo bar osnove BDE-a i termine koji se koriste u API-ju i help fajlu.

SAVET

BDE help fajl se instalira uz BDE i sada je povezan sa ostatkom Delphi help fajla. Ukoliko ga ne možete pronaći, možete ga potražiti u direktorijumu Program Files/Borland/Common Files/BDE (ili u direktorijumu u kom ste instalirali BDE). ■

Svaka aplikacija koja pristupa BDE-u smatra se klijentom i ima odvojenu vezu sa BDE-om. Ova veza se često naziva *sesijom*; globalna komponenta `Session` unutar VCL-a Vam obezbeđuje unapred određenu vezu, tako da u opštem slučaju o ovome ne morate brinuti. Globalna komponenta `Session` poziva funkciju `DbiInit` da bi ostvarila vezu.

BDE obrađuje zahteve svakog klijenta, svake sesije, koristeći odvojene kontekste. Jedna aplikacija može zahtevati nezavisne veze upotrebom više `TSession` komponenata. Ovo je obično potrebno za višelinijski pristup bazi podataka, kao što ćete videti u Poglavlju 16, jer ukoliko BDE ne kreira odvojene kontekste, može nastati zabuna kada dobije dva odvojena zahteva (što se može dogoditi samo kod višelinijskih aplikacija).

Kod BDE poziva niskog nivoa, neophodno je nekoliko parametara (u obliku korisničkih obrada koje nemaju veze sa Windows obradama). Evo kratke liste:

- Hendl baze podataka (`HDBIDB`) je hendl aktuelne baze podataka sa kojom aplikacija radi. Vrednost ovog hendla možete dobiti upotrebom svojstva `DBHandle` komponenata skupa podataka ili svojstva `Handle` komponente `TDatabase`.
- Hendl kursora (`HDBICur`) je hendl na rezultujući skup, pogled na podatke tabele ili upita. Kao što termin *kursor* naznačava, Vi možete da pretražujete podatke u ovom pogledu jer pristup podacima preko kursora znači da pristupate slogu. Vrednost ovog hendla možete dobiti upotrebom svojstva `Handle` komponente Table ili Query.

NAPOMENA

Upotreba kursora za pristupanje podacima baze podataka je tipična za SQL servere. BDE takođe primenjuje istu tehniku za Paradox i druge lokalne tabele da bi se obezbedio jedinstven način za pristupanje svim bazama podataka. Paradox, dBase i drugi lokalni formati namenjeni su prstupu okrenutom slogovima. ■

Pakovanje lokalne tabele

Jednostavan i uobičajen primer direktne upotrebe lokalnih BDE poziva je pakovanje tabele, operacija koja fizički uklanja obrisane slogove. Delphi Table komponenta nema ugrađenu ovu funkciju, verovatno zato što je potrebna samo za neke lokalne baze podataka (nema smisla u klijent/server okruženju).

Pakovanje tabela je veoma važno za dBase, gde se uklonjeni slogovi čuvaju unutar tabele sve dok se ona ne spakuje. Kod Paradoxa je ovo manje važno, jer mehanizam baze podataka može

ponovo upotrebiti fizičke lokacije obrisanih slogova za nove slogove. Da biste spakovali tabelu dBase, jednostavno upotrebite funkciju `DbiPackTable`, kao što je upotrebljena u narednom kodu, izdvojena procedurom `PackDBaseTable` primera DbPack:

```
Table.Close;
// reopen in exclusive mode
Table.Exclusive := True;
Table.Open;
// pack the table
Check (DBIPackTable (Table.DBHandle,
Table.Handle, nil, nil, True));
// remove the exclusive mode
Table.Close;
Table.Exclusive := False;
```

Kao što iz koda možete videti, pre poziva ove funkcije potrebno je da tabelu otvorite u ekskluzivnom modu, za šta je, možda, potrebno da je prethodno zatvorite. Kao alternativu prosleđivanju hendla kursora kao drugog parametra, možete ga podesiti na `nil` i proslediti naziv tabele kao treći parametar i konstantu `szDBase` kao četvrti parametar.

NAPOMENA

Da biste pozivali BDE funkcije iz programa potrebno je da dodate iskaz `uses` koji se referiše na BDE jedinicu. Delphi i dalje obezbeđuje alijase za BDE, te ukoliko Vam je potrebna Delphi 1 kompatibilnost, možete se referisati na jedinice DbiTypes, DbiProcs i DbiErrs. ■

Dok BDE obezbeđuje specifičnu funkciju za pakovanje tabele dBase, ne postoji odgovarajuća funkcija za Paradox fajlove. Alternativa je rekonstrukcija tabele; na ovaj način primoravate BDE da ažurira podatke, uklanjajući slogove koji su označeni za brisanje. Ova operacija rekonstrukcije može da se obavi funkcijom `DbiDoReconstructure`, koja je složena za upotrebu, jer je to generička, višenamenska funkcija.

Funkcija za parametre zahteva jedan ili više deskriptora tabele, tipa `CRTblDesc` (koji se prosleđuje kao treći parametar), i broj deskriptora (koji se prosleđuje kao drugi parametar). Evo primera koda iz procedure `PackPdoxTable` primera DbPack koji koristi `DbiDoReconstructure`:

```
var
  TableDesc: CRTblDesc;
  hDatabase: hDbiDB;
begin
  // get the database handle and close the table
  hDatabase := Table.DBHandle;
  Table.Close;
  // fill the table descriptor
  FillChar (TableDesc, SizeOf (CRTblDesc), 0);
with TableDesc do
begin
    StrPCopy (szTblName, table.TableName);
    StrPCopy (szTblType, szParadox);
    bPack := True;
end;
// restructure the table, packing it
if hDatabase <> nil  then
```

```
Check (DBIDoReconstructure (hDatabase, 1,
  @TableDesc, nil, nil, nil, False));
```

Kao što možete videti, većina polja deskriptora tabele i većina parametara funkcije zapravo se ne koriste; prava operacija rekonstrukcije bi zahtevala mnogo više koda. Jedina bitna stvar u prethodnom listignu je određivanje vrednosti parametra `True bPack` deskriptora tabele da bi se zahtevala operacija pakovanja prilikom operacije rekonstruisanja.

Kompletan izvorni kod dve rutine koje sam razmatrao možete pronaći u primeru DbPack. Program jednostavno prikazuje sve Paradox i dBase tabele datog alijasa (kao što možete videti na slici 10.23) i omogućava korisniku da selektuje tabelu i da je spakuje.

SLIKA 10.23 *Aplikacija DbPack Vam omogućava da spakujete Paradox i dBase tabele*

Upotreba Paradox fajlova na mreži

Kada želite da delite podatke baze podataka između više korisnika, potrebno je da deljene fajlove baze podataka čuvate na disku kojem se može pristupiti sa svakog kompjutera. Program se može nalaziti na mrežnom uređaju, ali se izvršava na lokalnoj mašini, kao i mehanizam baze podataka. Možete generisati jednostavan program za instaliranje za Vašu aplikaciju, upotrebom kopije InstallShield Expressa koji Vam je na raspolaganju u Delphi Professional i Client/Server CD-ovima. U svakom slučaju, slobodno možete da distribuirate BDE fajlove, koji su Vam na raspolganju i kao kompresovani CAB fajlovi za Internet distribuciju.

Postoji nekoliko BDE podešavanja koja treba da proverite kada delite Paradox fajlove baze podataka na mreži, kada je povezano više klijent aplikacija. Evo liste najvažnijih:

- Za LOCAL SHARE odredite TRUE. Ovo se može obaviti tako što se koristi BDE Configuration Utility, na strani System. Ova opcija je neophodna samo ukoliko iste fajlove koriste i aplikacije koje nisu zasnovane na BDE-u i ukoliko ne koristite Novell File Server.
- Koristite deljeni mrežni direktorijum. Ovo je vrednost alijasa u BDE Configuration Utilityju ili privremeni alijas komponente Database. Veoma je važno primetiti da mrežni uređaj koji koristite *mora* biti mapiran istom oznakom uređaja na svim klijent mašinama; u suprotnom, BDE kontrola se može "zbuniti" oko tabela koje aplikacija deli. Zapravo, kontrole se, preko naziva uređaja, referišu na tabele koje koriste.

- Takođe, koristite deljeni mrežni direktorijum za mrežni kontrolni fajl Paradox.NET. Ovo je vrednost parametra Net Dir koji možete podesiti upotrebom BDE pomoćnog programa za konfigurisanje ili svojstva `NetFileDir` komponente Session. Ovaj kontrolni fajl sadrži informacije o zaključavanju, sprečavanju više korisnika da izmene isti slog u isto vreme, što ćemo razmatrati u narednom odeljku ovog poglavlja.

UPOZORENJE

U slučaju grešaka dobićete zbunjujuću poruku Directory is controlled by other .NET file ili Multiple .NET files in use. ■

Upotrebom ovih vrednosti sve bi trebalo da funkcioniše, izuzev ukoliko program nije pravilno zatvoren u slučaju sistemske greške. Postoji nekoliko saveta koji bi trebalo da Vašu aplikaciju učine robusnijom:

- BDE funkcija `DBISaveChanges` čuva sve bafere mehanizma baze podataka na disku. Funkcija je korisna za lokalne tabele, ali je vitalna na mreži. Tipičan pristup je pozivanje ove funkcije iz događaja `AfterPost` tabele ili poruke `OnIdle` globalnog objekta `Application`.
- Kada postoji greška u indeksu Paradox tabele, možete jednostavno obrisati indeks fajl i ponovo ga izgraditi. Ovo obično rešava problem.
- Možete onemogućiti keširanje diska na kojem se nalaze fajlovi baze podataka, tako da se podaci odmah zapisuju na disk i ne gube se u slučaju sistemske greške.
- Konačno, možete upotrebiti specijalni pomoćni program za reparaciju tabele koji obezbeđuje Borland, a zove se TUtil32.DLL za 32-bitne verzije BDE-a. Postoje Delphi komponente koje pozivaju funkcije ovog DLL-a koji omogućava pokušaj popravke Paradox fajla sa tabelom.

Kontrola konkurentnosti

Ne možete ništa učiniti povodom toga: kada postoji više korisnika, oni će u nekom trenutku pokušati da istovremeno promene istu informaciju, što će prouzrokovati konflikt prilikom ažuriranja. Različiti serveri baza podataka primenjuju različite pristupe da bi obradili konkurentan pristup istim podacima. Postoje dva osnovna pristupa, jedan koji postoji u Pradoxu i drugi kod SQL servera:

- Paradox koristi pesimistički pristup. Kada korisnik prebaci slog u mod za izmene, slog se zaključava. Ostali korisnici mogu pročitati njegove vrednosti, ali isti slog ne mogu prebaciti u mod za izmene.
- Većina SQL servera baza podataka koristi optimistički pristup. Oni omogućavaju korisnicima da istovremeno edituju iste podatke i omogućavaju aplikacijama da pošalju prvobitne i izmenjene podatke, tako da korisnici mogu proveriti da li je neko drugi načinio izmene. Ovu tehniku ću obraditi u narednom odeljku (u delu koji je posvećen komponenti `TUpdateSql`).

Kada koristite Paradox u mrežnom okruženju, informacije o zaključavanju za svakog korisnika čuvaju se u deljenom Paradox.Net fajlu (što sam pomenuo u prethodnom odeljku). Čak i ako nemate mrežu, ovo možete proveriti pokretanjem iste aplikacije dva puta i pokušajem da editujete isti slog u oba prozora. Kako imate dve odvojene sesije baze podataka, po jednu za svaku instancu aplikacije, situacija je veoma slična kao kada imate dva korisnika koja pristupaju bazi podataka. Na slici 10.24 je primer poruke o grešci prilikom pokušaja editovanja istog sloga iz dve kopije programa.

SLIKA 10.24 *Poruka o grešci koju prikazuje Delphi aplikacija kada pokušate da izmenite slog koji je zaključao neki drugi korisnik ili program*

Postoji nekoliko stvari koje možete učiniti da biste imali više kontrole nad statusom sloga, bilo da želite da izbegnete ovakve vrste grešaka ili da jednostavno poboljšate izlaz ka korisniku. Ukoliko koristite eksplicitne pozive `Edit` metodu tabele, veoma jednostavno rešenje je da te pozive umetnete u `try-except` blok. Na ovaj način, ukoliko je slog zaključan i sistem se pozove na izuzetak, možete upozoriti korisnika da neko drugi pokušava da izmeni podatke, i da pitate da li da ponovo pokušate da obavite operaciju editovanja (posle nekog vremena) ili da je poništite.

Problem je u tome što imate veoma malo kontrole nad dužinom trajanja operacije editovanja, te ne znate koliko dugo je potrebno čekati pre nego što možete ponovo pokušati da izmenite slog koji je drugi korisnik zaključao. Drugi korisnik je mogao da ode na šoljicu kafe, a da pre toga nije poslao izmene. Postoji nekoliko tehnika koje možete upotrebiti da izbegnete ili smanjite ovakve situacije:

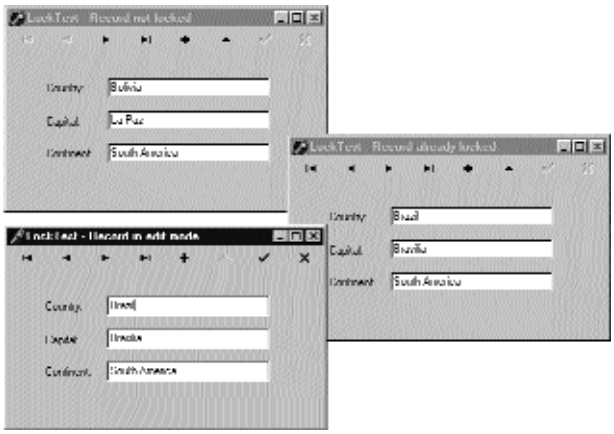
- Možete koristiti kontrole koje ne prepoznaju podatke, i operacije ažuriranja obradite kodom, čineći ih izuzetno brzim (kao što je pokazano primerom NonAware u prethodnom poglavlju).
- Možete zahtevati time-out nad operacijama editovanja (koristeći kontrolu timer), te ukoliko korisnik ne pošalje izmene posle nekog određenog vremena i ukoliko ne radi više sa aplikacijom, operacije editovanja tabele se automatski prekidaju. Ukoliko ovo učinite, trebalo bi da privremene izmene sačuvate u lokalnom baferu, da bi korisnik mogao da započne operaciju editovanja iz stanja u kome je prekinut. U suprotnom, korisnik bi ponovo morao da unosi podatke, jer ukidanje operacije editovanja, kod kontrola koje ne prepoznaju podatke, dovodi ažuriranja kontrola da bi se prikazali aktuelni podaci baze podataka.

U nekim slučajevima će Vam možda biti potreban status sloga, testirajući da li je zaključan (na primer, pre ažuriranja ili kada koristite kontrole koje ne prepoznaju podatke). Da biste proverili da li je još jedna tabela otvorena nad istim slogom, možete upotrebiti funkciju `DbiIsRecordLocked`. Kada je u pitanju više korisnika, ova funkcija ne pomaže jer proverava samo trenutnu sesiju. Zapravo, ne postoji BDE funkcija koja testira da li je neki drugi korsnik zaključao slog.

Ono što možete učiniti je da oponašate operaciju koju Delphi obavlja kada je tabela u modu editovanja. U tom slučaju VCL zaključava slog za pisanje. Obavljanje ove operacije nad tabelom (zapravo, nad kursorom) može promeniti status. Zbog toga je bolje kreirati klon kursora pa tek onda primeniti funkciju:

```
function IsRecordLocked (Table: TTable): boolean;
var
  Locked: BOOL;
  hCur: hDBICur;
  rslt: DBlResult;
begin
  Table.UpdateCursorPos
  // test if the record is locked by the current session
  Check (DbiIsRecordLocked (Table.Handle, Locked));
  Result := Locked;
  // otherwise check all sessions
  if (Result = False) then
  begin
    // get a new cursor to the same record
    Check (DbiCloneCursor (Table.Handle, False, False, hCur));
    try
      // try to place a write lock in the record
      rslt := DbiGetRecord (hCur, dbiWRITELOCK, nil, nil);
      // don't call Check: we want to do special actions
      // instead of raising an exception
      if rslt <> DBIERR_NONE then
      begin
        // if a lock error occured
        if HiByte (rslt) = ERRCAT_LOCKCONFLICT then
          Result := True
        else
          // if some other error happened
          Check (rslt); // raise the exception
      end
      else
        // if the function was successful, release the lock
        Check (DbiRelRecordLOck (hCur, False));
    finally
      // close the cloned cursor
      Check (DbiCloseCursor (hCur));
    end;
  end;
end;
```

Ova funkcija se koristi u primeru TestLock, veoma jednostavnom programu koji bi trebalo da testirate izvršavanjem više kopija. Program koristi tajmer i `OnDataChange` događaj izvora podataka da bi testirao status zaključavanja. U zaglavlju formulara on prikazuje da li se slog nalazi u modu za izmene (kada ga zaključavamo), da li je zaključan nekom drugom instancom aplikacije ili je dostupan. Tri kopije koje se izvršavaju u trima rzličitim situacijama možete videti na slici 10.25.



SLIKA 10.25 *Tri različita stanja primera TestLock: mod za izmene (zaključavanje aktuelnog programa), zaključavanje nekom drugom instancom ili korinikom, i stanje kada slog nije zaključan*

Pored prikazivanja informacija u zaglavlju, program onemogućava tri DBEdit kontrole svaki put kada neki drugi korisnik obavi zaključavanje:

```
procedure TNavigForm.TestLockStatus
begin
  // if the table is not in edit mode
  if Table1.State in [dsEdit, dsInsert] then
    Caption := 'LockTest - Record in edit mode'
  else if IsRecordLocked (Table1) then
  begin
    DbEdit1.ReadOnly := True;
    DbEdit2.ReadOnly := True;
    DbEdit3.ReadOnly := True;
    Caption := 'LockTest - Record already locked';
  end
  else
  begin
    DbEdit1.ReadOnly := False;
    DbEdit2.ReadOnly := False;
    DbEdit3.ReadOnly := False;
    Caption := 'LockTest - Record not locked'
  end;
end;
```

Transakcije baze podataka

Bilo da radite sa SQL serverom ili lokalnom bazom podataka, možete koristiti *transakcije* da bi Vaše aplikacije bile robusnije. Ideja transakcije se može opisati kao niz operacija koje se tretiraju kao jedna celina koja se ne može deliti.

Primer Vam može pomoći pri razjašnjavanju koncepta. Pretpostavimo da je potrebno da podignete platu svakom zaposlenom za određeni iznos, kao što smo to učinili u primeru Total u

prethodnom poglavlju. Ukoliko se tokom operacije desi greška, možda želite da poništite prethodne izmene. Ukoliko operaciju "podići platu svakog zaposlenog" posmatrate kao jednu transakciju, ona se mora obaviti u potpunosti ili potpuno ignorisati, ili razmotrite analogiju sa finansijskim transakcijama — ukoliko se obavi samo deo operacije, zbog greške, na kraju ćete možda imati različite pozajmice ili ćete imati višak novca.

Ima smisla tretirati operacije baze podataka kao transakcije. Možete započeti transakciju i obaviti nekoliko operacija koje sve treba posmatrati kao deo jedne velike operacije; zatim, na kraju, možete sačuvati izmene (commit) ili ih poništiti (rollback). Obično se izmene poništavaju kada se desi greška tokom operacija.

Rukovanje transakcijama u Delphiju je veoma jednostavno. Po definiciji se svaka edit/post operacija smatra transakcijom, ali ovakvo ponašanje možete izmeniti eksplicitnim rukovanjem. Jednostavno dodajte komponentu Database formularu ili modulu podataka, povežite svaku tabelu ili upit sa formularom ili modul podataka sa komponentom Database, a zatim upotrebite sledeća tri metoda TDatabase klase:

- `StartTransaction` označava početak transakcije.
- `Commit` potvrđuje sva ažuriranja nad bazom podataka koja su obavljena tokom transakcije.
- `Rollback` vraća bazu podataka u stanje pre započinjanja transakcije.

Jednostavan primer transakcija

Da bih Vam pokazao primer obrade transakcije koja se odnosi na Paradox fajlove, ja sam uneo tri metoda u korisničke operacije. Ovo nije standardna upotreba transakcija, ali bi trebalo da Vam pomogne da razumete kako funkcionišu. U narednom poglavlju ćemo videti nešto složenije primere u klijent/server okruženju.

Formular koji prikazuje primer Transact sadrži tri kontrole na panelu i tabelu koja je povezana sa veoma jednostavnim upitom, koji se veoma brzo povezuje sa komponentom baze podataka. Jednostavno postavite komponentu baze podataka na formular, dodelite vrednost svojstvu `DatabaseName` (koje se razlikuje od svojstva `Name`, koje predstavlja naziv komponente), a zatim odaberite ovaj naziv baza podataka za svojstvo `DatabaseName` komponente Query. Kao rezime, evo dela DFM fajla primera Transact:

```
object Databaset: TDatabase
  AliasName = 'DBDEMOS'
  Connected = True
  DatabaseName = 'MyData'
  SessionName = 'Default'
  TransIsolation = tiDirtyRead
end
object Query1: TQuery
  BeforeEdit = Query1BeforeEdit
  DatabaseName ='MyData'
  RequestLive = True
  SQL.Strings = (
    'select *from Employee')
end
```

Sada se možemo posvetiti kodu primera, koji je veoma jednostavan. Kada se klikne prva kontrola, program započinje transakciju (pozivajući `Database1.StartTransaction`) i aktivira druge dve kontrole. Kontrola Commit jednostavno poziva odgovarajući metod objekta baze podataka, `Database1.Commit`, posle prosleđivanja izmena, a zatim aktivira i deaktivira kontrole. Poslednja kontrola, Rollback, takođe treba da ažurira sadržaj kontrole DBGrid, pozivom metoda `Rferesh` komponente Query posle poziva `Query1.Cancel` i `Database1.Rollback`.

Ukoliko pokušate da pokrenete ovaj program, videćete da možete bazi podataka da pošaljete neke izmene, moguće i da promenite nekoliko slogova i da zatim poništite te izmene kada kliknete kontrolu Rollback. Nije potrebno da kliknete kontrolu Start jer se njen kod automatski izvršava svaki put kada započnete operaciju izmena:

```
procedure TForm1.Query1BeforeEdit(DataSet: TDataSet);
begin
  // start a transaction, if not already started
  if not Database1.InTransaction then
    BtnStartClick (self);
end;
```

Primetićete da se ovaj kod izvršava pre nego što se skup podataka prebaci u mod za izmene. Efekat rollback akcije možete videti na slici 10.26. Imajte na umu da postoji samo jedan nivo transakcija, ali za više transakcija nad različitim tabelama možete koristiti više komponenata baze podataka.

SLIKA 10.26 *Izlaz primera Transact pre (levo) i posle (desno) rollback operacije*

Transakcije se mogu koristiti nad Paradox fajlovima, ali samo nad tabelama koje imaju indeks; BDE rukuje transakcijom tako što zaključava sve odgovarajuće slogove. BDE može ostati bez resursa prilikom zaključavanja. Zbog toga bi trebalo da transakcije budu ograničene samo na nevizuelne operacije, kao što je slanje niza izmena jednoj ili više tabela. Prebacivanje novca sa jednog računa na drugi i povećanje plata zaposlenih su dobri primeri. Transakcije možete upotrebiti i za druge lokalne formate fajlova, recimo dBase i FoxPro.

Upotreba keširanih ažuriranja kao transakcija

Alternativa upotrebi transakcija nad lokalnim fajovima je upotreba keširanih ažuriranja. Šta je keširano ažuriranje? To je kada sve izmene čuvate u memoriji i zatim ih sve pošaljete tabeli. Ovo se dešava kada ažurirate pozivajući metod `ApplyUpdates` skupa podataka (prilikom ažuriranja jedne tabele), ili kada pozivate isti metod baze podataka (kada ažurirate više tabela odjednom). Ne zaboravite da primenite ažuriranja kada koristite keširana ažuriranja, ili će Vaše izmene biti izgubljene!

Keširana ažuriranja su slična transakcijama jer ih možete poništiti baš kao i transakcije, odbacujući lokalne izmene koje je korisnik načinio. Da bih Vam pokazao primer ovakvog pristupa, ja sam transformisao aplikaciju Transact u novi program koji sam nazvao CacheUpd. Program ne sadrži komponentu baze podataka, a za svojstvo `CachedUpdates` komponete Query sam odabrao vrednost `True`:

```
object Query1: TQuery
  CachedUpdates = True
  AfterPost = Query1Afterpost
  OnUpdateError = Query1UpdateError
  DatabaseName = 'DBDEMOS'
  RequestLive = True
  SQL.Strings (
    'select * from Employee')
end
```

Dve kontrole pozivaju metode `ApplyUpdates` i `CancelUpdates`, a zatim deaktiviraju obe kontrole. Pošto primenite ažuriranja, trebalo bi i da ih potvrdite, da osvežite keš, a takođe bi trebalo da zaustavite poruke o greškama jer njih obrađujemo odvojeno:

```
procedure TForm1.BtnApplyClick(Sender: TObject);
begin
  try
    // apply the updates and empty the cache
    Query1.ApplyUpdates;
    Query1.CommitUpdates;
    // set buttons
    BtnApplyEnabled := False;
    BtnCancel.Enabled := False;
  except;
    // silent exception
  end;
end;
```

Kada se prva izmena prosledi lokalnom kešu, obrada `AfterPost` događaja ponovo aktivira kontrole. Ukoliko korisnik izađe iz aplikacije dok još uvek postoje izmene koje čekaju da se primene, mi tražimo konfirmaciju:

```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  // if there are pending changes, ask the user what to do
  if Query1.UpdatesPending and
      (MessageDlg ('Apply the pending updates?',
      mtConfirmation, [mbYes, mbNo], 0) = mrYes) then
    Query1.ApplyUpdates;
end;
```

Kada korisnik izmeni slog, BDE stavlja uobičajeno zaključavanje, ali odmah uklanja zaključavanje kada se izmene pošalju u lokalni keš. Zbog toga dva korisnika mogu imati izmene u lokalnoj memoriji. Prvi korisnik koji fizički primenjuje izmene, šalje podatke bazi podataka, dok je drugi onemogućen greškom koja označava da drugi korisnik menja podatke. U ovom slučaju Delphi poziva specifični događaj `OnUpdateError` skupa podataka koji obradi prosleđuje nekoliko parametara:

```
procedure Query1UpdateError (DataSet: TDataSet;
  E: EDatabaseError; UpdateKind: TUpdateKind;
  var UpdateAction: TUpdateAction);
```

Prvi parametar je skup podataka, drugi je greška koja će se prikazati korisniku, treći je opis ažuriranja (koji može biti `ukModify`, `ukInsert` i `ukDelete`), a poslednji je akcija koju želite da obavite (unapred određena vrednost `uaFail`, `uaAbort`, `uaSkip`, `uaRetry` ili `uaApplied`). Program može zaključati podatke koji su vezani za grešku i odrediti koja je operacija obavljena, mada je teško ispraviti problem. Sve što možemo uraditi jeste da upotrebimo svojstva `OldValue` i `NewValue` svakog polja da bismo odredili uslove greške, i funkciju `UpdateStatus` za određivanje operacije ažuriranja (izmena, umetanje ili uklanjanje). Da bismo popravili grešku, možemo pravilno odrediti `NewValue` i pokušati da ponovo primenimo izmene, mada to može dovesti do neprekidne petlje.

U primeru CacheUpd ja sam koristio funkciju `UpdateStatus` da bih prikazao status svakog sloga na statusnoj liniji kako se korisnik pomera preko slogova:

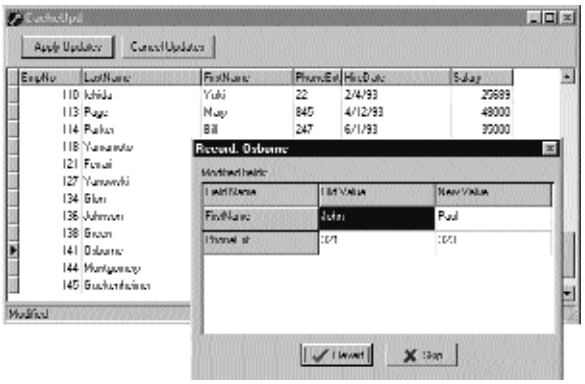
```
procedure TForm1.Query1AfterScroll (DataSet: TDataSet);
begin
  // show the record update status in the status bar
  case Query1.UpdateStatuus of
    usUnmodified:
      StatusBar1.SimpleText := 'Non Modified';
    usModifled:
      StatusBar1.SimpleText := 'Modified';
    usInserted:
      StatusBarl.SimpleText := 'Inserted';
  end;
end;
```

Kada se dogodi greška, program prikazuje sekundarni složeni formular koji u tabeli prikazuje sva polja sloga koje je izmenio korisnik. Tabela je deo jednostavnog okvira za dijalog koji se kreira u vreme izvršavanja i inicijalizuje metod `FormCreate` ispunjavajući prvi red opisom svake kolone. Međutim, prava inicijalizacija se dešava kada se desi greška. Evo dugačkog, ali potpunog, listinga događaja `OnUpdateError` upita:

```
procedure TForm1.Query1UpdateError(DataSet: TDataSet;
  E: EDatabaseError; UpdateKind: TUpdateFifld;
  var UpdateAction: TUpdateAction);
var
  strDescr: string;
  I, nRow: Integer;
begin
  nRow := 0;
  // create the dialog box
  ErrorsForm := TErrorsForm.Create (nil);
  try
```

```
    // set the caption to a description of the record
    ErrorsForm.Caption := 'Record:  ' +
      Dataset. FleldByName( 'LastName') AsString;

    // for each modified field
    for I := 0 to DataSet.FieldCount - 1 do
      if DataSet.Fields [I] .OldValue <>
        DataSet.Fields [I].NewValue then
      begin
        // add a row to the string grid
        Inc (nRow);
        ErrorsForm.StringGrid1.RowCount := nRow + 1;
        // copy the data to the new row
        with ErrorsForm.StringGrid1, Oataset.Fields[I] do
        begin
          Cells [0, nRow] := FieldName;
          Cells [1, nRow] := string (OldValue);
          Cells [2, nRow] := string (NewValue);
        end;
      end;

  // if new items were added, show the dialog
  if (nRow > 0) and (
    ErrorsForm.ShowModal = mrok) then
  begin
      // revert the record and hide the message
      (DataSet as TQuery).RevertRecord;
      UpdateAction := uaAbort
    end
    else
      // skip the record, keeping it in the cache
      UpdateAction := uaSkip;
  finally
    ErrorsForm. Free;
  end;
end;
```

Efekat ovog programa je vidljiv samo kada pokrenete dve kopije, učinite izmene nad istim slogom u oba prozora, a zatim primenite izmene. Ne primenjujte izmene iz jednog prozora pre nego što načinite izmene u drugom prozoru, inače će sistem automatski ažurirati vrednost kada započne operacija editovanja. Primer sekundarnog okvira za dijalog u slučaju greške možete videti na slici 10.27.



SLIKA 10.27 *Okvir za dijalog sa greškom koji se prikazuje kada postoji konflikt u ažuriranju u primeru CacheUpd. Primetićete opis sloga na statusnoj liniji glavnog formulara.*

SAVET

Ukoliko se u Vašim tabelama koristi ID za identifikovanje slogova, upotreba keširanih ažuriranja Vas može dovesti u nepriliku, jer različite aplikacije imaju različite poglede na podatke (prema onome što imaju u memoriji, a ne prema onome što je na disku). Uobičajeno rešenje je imati odvojenu tabelu sa brojačem. Ovo je tipična tehnika kod server/klijent aplikacija. ■

Šta je sledeće?

U ovom poglavlju smo upoznali mnoge napredne karakteristike Delphi programiranja baza podataka, koje se odnose kako na lokalne baze podataka tako i na SQL servere. Upoznali smo se sa strukturom aplikacija za baze podataka koje se zasnivaju na više formulara, upotrebom modula podataka i Data Dictionary pri izradi složenih aplikacija, a izradili smo i MDI program.

Pošto mnogi Delphi programeri često koriste Paradox fajlove, posvetili smo dosta prostora upotrebi ovih fajlova u višekorisničkom mrežnom okruženju, ali i obradi grešaka, konkurentnosti i drugim temama.

Ipak, Access tabele su sve popularnije kao lokalni fajlovi, a Delphi 5 podrška za ADO čini jednostavnom upotrebu Access i drugih baza podataka. To će biti tema Poglavlja 12 u kojem ćemo koristiti komponente za pristup podacima koje ne zahtevaju BDE. Komponente ovog tipa, InterBase komponente, biće razmatrane u narednom poglavlju uz detaljan uvod u klijent/server programiranje u Delphiju.

POGLAVLJE

11

Klijent/server programiranje

U poslednja dva poglavlja smo se upoznali sa Delphi podrškom programiranju baza podataka i posvetili smo posebnu pažnju upotrebi lokalnih fajlova (naročito Paradox fajlova) koji mogu i ne moraju biti deljeni na mreži. Ovo poglavlje se odnosi na upotrebu SQL server baza podataka, a posebna pažnja je posvećena klijent/server programiranju. Samo jedno poglavlje ne može biti dovoljno da bi se ova složena tema detaljno obradila, te ću ja samo dati uvod iz perspektive Delphi programera i navešću neke savete i uputstva. Naredno poglavlje će biti nastavak klijent/server programiranja sa podrškom za Microsoft ADO.

RDBMS (Relational DataBase Management System – relacioni sistem manipulisanja bazom podataka), ili SQL server koji ćemo koristiti, je InterBase jer je uključen u Client/Server verziju Delphija (Local verzija InterBasea je, takođe, na raspolaganju u verziji Delphi Professional). Ja neću procenjivati da li je ovaj server bolji ili robusniji od drugih servera (i često mnogo skuplji), već ću jednostavno koristiti InterBase za testiranje delova koda i primera koje dajem u tekstu.

Sa alatima za brzi razvoj aplikacija (RAD) kao što je Delphi, Vi zaista možete da upotrebite neke komponente i kod koji je načinjen za aplikacije za lokalne baze podataka, i iskoristite ih u klijent/server okruženju. Ipak, ova mogućnost se može pokazati opasnom ukoliko ste početnik, jer standardna tehnika koja funkcioniše za lokalni pristup, može biti neverovatno neefikasna u kljent/server aplikacijama.

NAPOMENA

Većina informacija u ovom poglavlju odnosi se samo na Enterprise izdanje (nekada Client/Server) Delphija. Naravno, opšte informacije o SQL-u su korisne kada god pišete upite nad lokalnim fajlovima, a osnovni koncepti klijent/server arhitekture se mogu primeniti čak i kada serveru pristupate preko ODBC-a ili upotrebom drugih tehnika koje su Vam na raspolaganju u izdanju Delphi Professional. ■

Struktura klijent/server programiranja

Aplikacije za baze podataka iz prethodnog poglavlja su koristile BDE za pristupanje podacima koji su se čuvali u fajlovima bilo na lokalnoj mašini bilo na mrežnom kompjuteru. U oba slučaja smo koristili server fajlova, čija je jedina uloga čuvanje fajlova na hard disku, jer se mehanizam baze podataka (BDE) izvršavao isključivo na kompjuteru na kojem se nalazila aplikacija. U ovakvoj konfiguraciji, kada bismo postavili upit nad tabelom, njeni podaci bi se kopirali u lokalni keš BDE-a, a zatim obrađivali.

Kao primer, zamislite da radite sa tabelom kao što je tabela Employee (deo InterBase IBLocal baze podataka koju dobijate uz Delphi), dodajete hiljade slogova i smeštate je na mrežni kompjuter koji služi kao server fajlova. Kada bismo poželeli da saznamo kolika je najveća plata u kompaniji, mogli bismo da otvorimo komponentu Table (`ExmpTable`) i izvršimo sledeći kod:

```
EmpTable.Open;
EmpTable.First;
MaxSalary := 0;
while not EmpTable.Eof do
begin
  if EmpTable.FieldByName ('Salary').AsCurrency > MaxSalary then
    MaxSalary := EmpTable.FieldByName ('Salary').AsCurrency;
  EmpTable.Next;
end;
```

Efekat ovakvog pristupa je da premeštamo sve podatke (velike) tabele sa mrežnog kompjutera na lokalnu mašinu, što je operacija koja traje nekoliko minuta. Kako Delphi sadrži komponentu Query, mogli biste da upotrebite sledeći SQL kod da biste izračunali maksimalnu vrednost:

```
select Max(Salary) from Employee
```

U slučaju tabele Paradox, ovaj upit bi izvršio lokalni SQL mehanizam BDE-a, a čitav skup podataka tabele bi trebalo i dalje premestiti sa mrežnog kompjutera na lokalni, što bi, takođe, dalo loše performanse. Međutim, ukoliko koristite InterBase i prepustite serveru da izvrši SQL kod, samo je rezultujući skup — jedan broj — potrebno prebaciti na lokalni kompjuter.

NAPOMENA

Dva prethodna isečka koda su deo primera GetMax, koji sadrži kod kojim se upoređuje vreme izvršavanja dvaju pristupa. Upotreba komponente Table nad malom tabelom Employee izvršava se oko deset puta duže od upotrebe upita, čak i kada je InterBase server instaliran na lokalnom kompjuteru. ■

Ukoliko želite da sačuvate veliki broj podataka na centralnom kompjuteru i da izbegnete premeštanje podataka na klijent kompjutere radi obrade, jedino rešenje je da manipulaciju podacima prepustite centralnom kompjuteru i da klijentima šaljete samo ograničenu količinu informacija. Ovo su osnove klijent/server programiranja.

U opštem slučaju ćete koristiti postojeći program sa servera (RDBMS) i pisaćete klijent aplikacije koje su sa njim povezane. Ponekad ćete pisati i klijent i server aplikacije, kao što je slučaj kod višelinijskih aplikacija. Delphi podrška za ovakav tip programa — MIDAS arhitektura — je obrađena u Poglavlju 21.

Transfer podataka iz lokalnih fajlova do mehanizma baze podataka SQL servera (upsizing) obavlja se isključivo zbog performansi i da bi se omogućilo prenošenje velike količine podataka. Ukoliko se vratimo na prethodni primer, upit koji se u klijent/server okruženju koristi za određivanje najveće plate, bio bi izračunat od strane RDBMS-a, čime bi se klijent kompjuteru poslao samo konačni rezultat, jedan broj. Sa moćnim server kompjuterom (kao što je višeprocesorski Sun SparcStation), ukupno vreme za izračunavanje rezultata može biti minimalno.

Ipak, postoje i drugi razlozi zbog kojih treba odabrati klijent/server arhitekturu:

- **KOLIČINA PODATAKA** Paradox tabela ne može preći veličinu od 2GB, ali već i oko 300MB mogu početi da se javljaju ozbiljni problemi oko brzine, a greške sa indeksima postaju češće.
- **POTREBA ZA KONKURENTNIM PRISTUPOM PODACIMA** Paradox koristi Paradox.NET fajl u kojem se čuvaju informacije o tome koji korisnik pristupa određenim tabelama i slogovima. Paradox pristup za opsluživanje više korisnika koristi pesimističko zaključavanje (pessimistic locking). Kada korisnik započne operaciju editovanja nad slogom, ni jedan od ostalih korisnika ne može da započne editovanje (da bi se izbegli bilo kakvi konflikti ažuriranja), kao što smo videli u prethodnom poglavlju. U sistemu sa deset korisnika ovo može da dovede do ozbiljnih problema, jer jedan korisnik može da blokira rad mnogih korisnika. SQL server baze podataka nasuprot tome koriste optimističko zaključavanje (optimistic locking), pristup koji dozvoljava da više korisnika radi sa istim podacima i odlaže kontrolu konkurentnosti sve dok korisnik ne pošalje ažurirane podatke.
- **ZAŠTITA I SIGURNOST** RDBMS obično sadrži mnogo više mehanizama zaštite nego što može da pruži jedna lozinka nad Paradox tabelom. Kada se Vaša aplikacija bazira na fajlovima, zlonamerni ili nepažljivi korisnik može jednostavno da obriše te vitalne fajlove. Kada su SQL serveri bazirani na robusnim operativnim sistemima, oni obezbeđuju više nivoa zaštite, čine pravljenje rezervnih kopija lakšim i često samo administratoru baze podataka dozvoljavaju da izmeni strukturu tabela.

- **PROGRAMABILNOST (MOGUĆNOST PROGRAMIRANJA)** RDBMS može da sadrži radna pravila u formi uskladištenih procedura, okidača, pogleda nad tabelama i drugih tehnika koje ćemo razmatrati u ovom poglavlju. Glavna stvar kod klijent server programiranja je izbor kako podeliti kod između klijent i server aplikacije.
- **KONTROLA TRANSAKCIJA** U prethodnom poglavlju smo videli da Paradox i BDE nude ograničenu podršku transakcijama, ali je obično RDBMS podrška mnogo veća. To je još jedan važan aspekt za sveukupnu robusnost sistema.

Klijent/server i Delphi

Razmotrimo sada kako se Delphi uklapa u klijent/server sliku. Kako nam Delphi pomaže da izradimo klijent/server aplikaciju? Kao što sam pomenuo, još uvek možete koristiti sve komponente i tehnike koje ste videli u prethodna dva poglavlja, mada u nekim slučajevima alternativni pristupi mogu povećati moć RDBMS-a.

Komponenta Database

Kada su u pitanju lokalne aplikacije, programeri se obično na baze podataka referišu navođenjem alijasa putanje za svojstvo `DatabaseName` komponenata Table i Query. Alternativa je upotreba komponente Database za definisanje lokalnih alijasa i omogućavanje da se komponenta DataSet referiše na te lokalne alijase.

Kao primer, razmotrite komponente aplikacije GetMax koju smo ranije razmatrali:

```
object Database1: TDatabase
  AliasName = 'IBLOCAL'
  Connected = True
  DatabaseName = 'IB'
  LoginPrompt = False
  Params.Strings = (
    'USER NAME=SYSDBA'
    'PASSWORD=masterkey')
  SessionName = 'Default'
end
object EmpTable: TTable
  DatabaseName = 'IB'
  TableName = 'EMPLOYEE'
end
object EmpQuery: TQuery
  DatabaseName = 'IB'
  SQL.Strings = (
    'select Max(Salary) from Employee ')
end
```

Kada su u pitanju klijent/server aplikacije, upotreba komponente Database gotovo da je obavezna, jer je potrebna da bi se definisala konektivnost i parametri pristupa (korisničko ime i lozinka, kao što možete videti iz svojstva `Params`) i za rukovanje transakcijama.

Imajte na umu da komponenta Database uspostavlja vezu sa RDBMS-om predstavljajući jednog od klijenata sistema. Na većini servera ovo zahteva licencu, a Vaša organizacija obično plaća

ograničeni broj licenci. Ukoliko neke aplikacije ili kompjuteri koriste više konekcija sa serverom, mogu se računati kao više korisnika! Na sreću određivanjem svojstva `KeepConnection` komponente Database možete navesti da li da se konekcija sa bazom podataka održava aktivnom čak i kada nema aktivne komponente DataSet koja koristi konekciju. Ukoliko Vaš program može da dobije neke podatke i lokalno ih obradi, isključivanje sa servera može pomoći u očuvanju licence.

Uloga BDE-a

Kakva je uloga BDE-a u ovakvoj arhitekturi? Kada su u pitanju klijent/server aplikacije izrađene u Delphiju 4 ili ranijim verzijama, još uvek je potrebno da klijent programi imaju interakciju sa lokalnom kopijom BDE-a instaliranom na klijent mašini. Upotrebom novih komponenata Delphi 5 ADO ili InterBase ExPress možete izbeći instaliranje BDE-a na klijentu (ali, naravno, nećete moći da koristite njegove karakteristike).

Razmotrimo prvo tradicionalni pristup. (Još uvek će biti veoma čest i kada je u pitanju Delphi 5.) BDE ne zna kako da rukuje RDBMS-om; i dalje koristi drajvere, koji se nazivaju SQL Links, da bi obavio svoje operacije. Kao alternativu za to, BDE može komunicirati sa ODBC drajverima. Inprise obezbeđuje osnovne BDE drajvere za InterBase, Oracle, Informix, MS SQL, Sybase i DB2.

Ukoliko je BDE i dalje neophodan na lokalnim mašinama, on zapravo može biti veoma efikasan. Na primer, kada koristite pass-through mod upita, BDE ne pokušava da interpretira SQL, već ga direktno prosleđuje RDBMS serveru. Ovo Vam omogućava da koristite specifične SQL komande, a takođe ubrzava izvršavanje. Pass-through se aktivira upotrebom pomoćnog programa BDE Administrator.

Postojanje BDE-a između klijenta i servera može pomoći pri izradi aplikacija koje rade sa više servera. U praksi, ipak, to nije lako učiniti i istovremeno dobiti najbolje performanse, zbog razlika između SQL dijalekata sa kojima radi svaki SQL server. Tipovi podataka se drugačije obrađuju na različitim serverima. Ukoliko se ista tabela nalazi na dva servera koja imaju razlike u tipovima podataka, Delphi će morati da koristi dva različita `TField` objekta.

Takođe, imajte na umu da BDE tretira podatke pristupom orijentisanim ka slogovima lokalnih fajlova, a ne pristupom orijentisanim ka skupovima, kao što je slučaj sa SQL serverima.

SAVET

Interesantan aspekt BDE-a je njegova mogućnost izvršavanja heterogenih spajanja, to jest, može izvršiti iskaz SELECT nad više tabela različitih baza podataka (upotrebom različitih SQL servera i lokalnih tabela). Ovo može biti korisno jer mnogi serveri ne pružaju podršku povezivanju sa spoljašnjim tabelama, ali treba da imate na umu da DBE, da bi izvršio ovakvu operaciju, često treba da preuzme čitav sadržaj tabela koje se koriste u upitu koji se izvršava na klijent kompjuteru. ■

Postoje dve alternative upotrebi BDE-a: direktna upotreba API-ja određenog servera, kao u InterBase Express komponentama, ili upotreba različitih mehanizama baze podataka, kao kod kombinacije ADO sa OLE DB. Upotreba API servera može dovesti do boljih performansi, ali aplikacija neće biti prenosiva na drugi SQL server. Osnovni Delphi 5 skup komponenata za InterBase sigurno čini API server pristup privlačnim, i razmatraću ga kasnije u ovom poglavlju (posle uvoda u klijent/server programiranje upotrebom tradicionalnog pristupa i BDE-a).

Još jedna alternativa, koja je predstavljena u narednom poglavlju, jeste upotreba ADO-a umesto BDE-a. Kao što ćemo videti, prednost ADO-a je u tome što je mehanizam deo operativnog sistma Windows 2000, te ga možete uzeti zdravo za gotovo (ukoliko ne sada, onda u najbližoj budućnosti). Takođe, upotreba ADO-a može biti dobar izbor uz Microsoft tehnologiju baza podataka (uključujući MS Access i MS SQL Server).

Ukoliko planirate da koristite Oracle, smatram da su ADO i BDE ekvivalentno dobre alternative: mada budućnost ADO-a sigurno izgleda svetlija, BDE podrška za Object Relational model Oracle 8.0 verovatno je superiornija. Pošto se Oracle i Microsoft bore oko prevlasti nad bazama podataka (uključujući, na primer, SQL proširenja za objektni relacioni model), mogu se desiti iznenađenja u ovoj oblasti. Imajte na umu da ćete, ukoliko je Oracle Vaš konačni izbor, verovatno moći da koristite direktne komponente slične onima koje Delphi obezbeđuje za InterBase (posetite, na primer, Direct Oracle Access na adresi `www.allroundautomations.nl`).

Od lokalnog do klijent/server programiranja

Sada možemo da obratimo pažnju na određene tehnike koje su korisne za klijent/server programiranje. Imajte na umu da je osnovni cilj da pravilno distribuirate informacije između servera i klijenta i da smanjite mrežni protok koji je potreban za prebacivanje informacija.

Osnova ovog pristupa je dobro dizajnirana baza podataka, pod čime se podrazumeva kako struktura tabela, tako i odgovarajuća provera podataka i veza, ili radnih pravila. Primena provere podataka na serveru je veoma važna jer je integritet baze podataka jedan od ključnih ciljeva svakog programa. Ipak, klijent strana takođe treba da proverava podatke da bi se poboljšao korisnički interfejs i da bi se unos i obrada podataka učinili bližim korisniku. Ima malo smisla dopustiti korisniku da unese podatke i da zatim dobije poruku o grešci sa servera kada možemo da sprečimo pogrešan unos.

NAPOMENA

Ukoliko koristite alat CASE za definiciju baze podataka, ili kasnije uvezete definiciju u takav alat, možete upotrebiti Delphi Case Wizard da generišete odgovarajući Data Dictionary i da svi objekti polja kreirani u vreme dizajniranja automatski uvezu veze navedene na serveru. ■

Jednosmerni kursori

Kod lokalnih baza podataka, tabele su sekvencijalni fajlovi čiji je redosled ili fizički redosled, ili je definisan indeksom. Nasuprot tome, SQL serveri rade sa logičkim skupovima podataka koji nemaju veze sa fizičkim redosledom. *Relacioni* server baze podataka rukuje podacima prema relacionom modelu, matematičkom modelu zasnovanom na teoriji skupova.

Ono što je važno za trenutnu diskusiju jeste da se u relacionoj bazi podataka slogovi (ponekad se nazivaju tuples) označavaju ne preko pozicije, već isključivo preko primarnog ključa, koga može sačinjavati jedno ili više polja. Kada jednom dobijete skup slogova, server svakom od njih dodaje referencu na naredni, što čini brzim prelazak sa jednog sloga na naredni, ali čini neverovatno sporim prelazak na prethodni. Zbog toga se često kaže da RDBMS koristi jednosmerni kursor.

Povezivanje takve tabele ili upita sa kontrolom DBGrid čini veoma sporim prelazak na prethodni slog. BDE umnogome pomaže jer u lokalnom kešu čuva slogove koji su već učitani u tabelu. Zato, kada se pomerimo na naredne slogove, oni se zahtevaju sa SQL servera, ali kada se vraćamo unazad, uskače BDE i obezbeđuje nam podatke. Drugim rečima, BDE ove kursore čini potpuno dvosmernim, mada se može zauzeti nešto više memorije.

NAPOMENA

Jednostavan primer je DBGrid koji se koristi za pretraživanje cele tabele u lokalnim programima, ali ga treba izbegavati u klijent/server okruženju. Bolje je izdvojiti deo slogova i polja koja nas interesuju. Da li je potrebno da vidite spisak naziva? Zahtevajte sve koji počinju slovom A, zatim sve koji počinju slovom B, ili neka korisnik unese početno slovo. ■

Ukoliko prethodno vraćanje unazad može izazvati probleme, imajte na umu da je prelazak na poslednji slog još gori; ova operacija obično zahteva dobijanje svih slogova.

Slična situacija je i sa `RecordCount` svojstvom skupova podataka. Izračunavanje broja slogova često implicira njihovo prebacivanje na klijent kompjuter. To je razlog što kontrola vertikalnog klizača DBGrida funkcioniše za lokalne tabele, ali ne i za tabele koje su na serveru. Ukoliko je potrebno da znate broj slogova, pokrenite zaseban upit da biste prepustili serveru da izračuna broj slogova (ne radite to na klijentu). Na primer, možete videti koliko slogova će biti selektovano iz tabele Employee kada su Vam potrebni slogovi gde je plata veća od 50000:

```
select count(*)
from Employee
where Salary > 50000
```

SAVET

Upotreba SQL instrukcije `count(*)` je zgodan način za izračunavanje broja slogova koji će biti proizvedeni upitom. Umesto džoker karaktera *, mogli smo da upotrebimo naziv nekog polja, recimo, `count(First_Name)`, uz kombinaciju sa `distinct` ili `all`, da biste prebrojali samo slogove sa različitom vrednošću polja, ili sve slogove koji sadrže vrednost različitu od null. ■

Komponente Table i Query u Client/Server aplikaciji

U Delphiju postoje dve komponente koje možete koristiti za pristupanje postojećoj tabeli baze podataka. To su komponente Table i Query. Kada izrađuju klijent/server aplikacije, programeri češće koriste komponentu Query, ali to svakako nije obavezno, a postoje i slučajevi kada jednostavnija komponenta Table nema nedostatke. Ukratko navodim argumente za i protiv ovih komponenata:

- Komponentu Table ne treba koristiti za pristup velikim tabelama, ali za male tabele odlično funkcioniše. Otvaranjem komponente Table ne prebacujete sav sadržaj tabele na lokalnu mašinu; podaci se prebacuju tek kada pristupite nekom određenom slogu.
- Takođe, imajte na umu da kada radite sa komponentom Table, BDE od servera prvo zahteva strukturu tabele, a zatim podatke tabele. Ova dva koraka su neophodna za pravilno određivanje interne strukture BDE-a i oni se ne izvršavaju kada je u pitanju komponenta Query. Ukoliko aktivirate BDE karakteristiku

Schema Caching, logička struktura tabele će se čuvati lokalno. Naravno, ovo može stvoriti probleme ukoliko se logička struktura tabele promeni na serveru.

- Jedan problem sa komponentom Table je u tome da BDE oponaša dvosmerni kursor lokalnim keširanjem podataka. Kada je u pitanju komponenta Query, Vi možete, svojstvom `Unidirectional`, navesti da li želite ovakvo keširanje ili ne.
- Još jedna strana koju treba uzeti u obzir jeste da možete da editujete rezultate jednostavnih upita, šaljući podatke nazad na SQL server. Ovo se postiže određivanjem vrednosti `True` za svojstvo `RequestLive`. Za složenije SQL upite potrebno je da koristite komponentu UpdateSQL, koju ćemo razmatrati kasnije u ovom poglavlju.
- Kada pokušavate da minimizirate podatke koji se prebacuju između klijenta i servera, potrebno je da uzmete u obzir veličinu svakog sloga kao i ukupan broj slogova. Kada upitom selektujete samo nekoliko polja, u obzir se uzima samo deo podataka. Komponenta Table uvek pokušava da prebaci ceo slog na lokalnu mašinu, čak i ako izdvojite neka polja upotrebom Fields editora. Isti problem se pojavljuje kada imate aktivne upite (određivanjem svojstva `RequestLive`). U ovom slučaju, BDE-u je potrebno da vidi ceo slog da bi mogao da pošalje pravilne komande za ažuriranje. To znači da je selektovanje svih slogova tabele aktivnim upitom ekvivalentno komponenti Table.
- Komponenta Query nije ograničena samo na SQL iskaze `select`; možete je koristiti za umetanje i uklanjanje slogova. Kada komponenta Query vrati skup podataka, aktivirate je metodom Open (ili ekvivalentnom operacijom, određivanjem vrednosti `True` za svojstvo `Active`). Kada se komponenta Query koristi za izvršavanje operacije na serveru, aktivirate je pozivom metoda `ExecuteSQL`.

Parametarski upiti i Null vrednosti

Parametarski upiti su veoma korisna tehnika. U osnovi, oni omogućavaju izvršavanje više upita sa različitim skupom pravila, dok server treba samo jednom da načini strategiju za rešavanje upita.

Inicijalnu pripremu strategije pristupa upita možete zahtevati pozivom metoda Prepare komponente Query. Ovom operacijom server dobija upit, proverava sintaksu i, dok ga kompajlira, određuje kako će koristiti indekse i druge tehnike pristupa. Ukoliko se zatim upit izvršava više puta, biće brži jer su se ove inicijalne operacije već izvršile. Naravno, potrebno je da ponovo pozovete `Prepare` ukoliko promenite tekst SQL upita. Takođe, ne zaboravite poziv `Unprepare` na kraju da biste oslobodili BDE resurse.

Zapamtite da neki moćni SQL serveri mogu istu operaciju izvršiti keširanjem zahteva i automatski odrediti da isti zahtev šaljete dva puta. Ukoliko je server dovoljno "pametan", priprema upita može da dâ malo ili nikakvo poboljšanje.

NAPOMENA

Kada pišete parametarski upit za SQL server, morate pažljivo da uzmete u obzir i `null` vrednosti. Zapravo, da biste proverili `null` vrednost, ne smete da napišete `filed = null`, već upotrebite specifični izraz `field is null`. ■

Upotreba filtera Table i Query

Jedan način za ograničavanje količine podataka koje daje tabela, jeste filtriranje (izdvajanje) tabele. Upotrebom svojstva `Filter` komponente Table možete da naznačite uslov sličan `where` klauzuli upita. Kada radite sa lokalnim bazama podatka, filter primenjuje BDE, ali kada radite sa SQL serverom, BDE prosleđuje uslov serveru u upitu koji je generisan za tabelu. Ovo čini filtrirane tabele pokretnim između lokalnih i klijent/server aplikacija.

UPOZORENJE

Situacija je drugačija ukoliko slogove filtrirate Pascal kodom koristeći događaj `OnFilterRecord`. U tom sučaju svi slogovi se šalju klijent aplikaciji koja sama obavlja filtriranje. ■

Ukoliko filter koristite uz komponentu Query, operacija filtriranja će se uvek obaviti lokalno preko BDE-a, čak i kada radite sa SQL serverom. U ovom slučaju BDE od servera zahteva ceo rezultujući skup upita. Ovo je razumno samo kada korisnik aplikacije često menja uslove filtriranja. Za upit će se izmeniti samo lokalni filter i koristiće se samo podaci iz lokalnog keša. Za tablu će BDE generisati ažurirani upit koji treba izvršiti.

Upoznavanje sa Local InterBaseom

Sada kada smo se upoznali sa teorijom i nekim čestim zamkama klijent/server programiranja, možemo početi da proučavamo neke praktične primere koji nas upoznaju sa Local InterBaseom, jednokorisničkom verzijom Inprise RDBMS-a koji dobijate kao deo Professional i Enterprise izdanja Delphija. Ova lokalna verzija SQL server mehanizma je korisna za razvoj i testiranje klijent/server aplikacija na jednom kompjuteru.

NAPOMENA

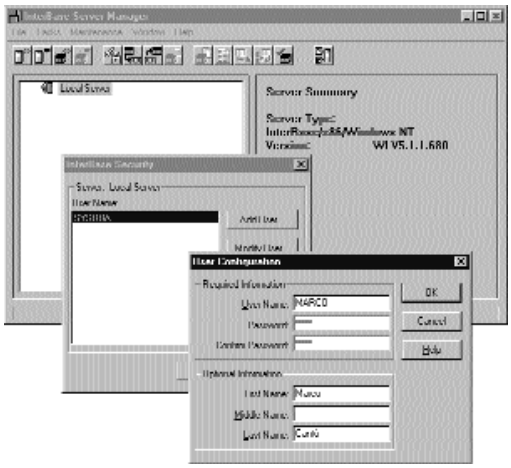
Među prednostima upotrebe InterBasea su i jednostavni alati za administraciju, koji su Windows aplikacije, i dobre performanse kada su u pitanju veliki skupovi podataka. ■

Posle instaliranja Local InterBasea (imate ga na CD-u Delphi 5), moći ćete da aktivirate server iz Windowsovog menija Start. (Program IBServer.EXE se nalazi u BIN direktorijumu InterBasea, obično kao poddirektorijum Program Files\Borland\InterBase ili Program Files\InterBase Corp\InterBase, što zavisi od instalacije.) Kada je server aktivan, videćete odgovarajuću ikonu u Tray Icon oblasti Windows TaskBara. Meni koji je povezan sa ovom ikonom omogućava Vam da konfigurišete InterBase i aktivirate njegovo automatsko pokretanje. Kada dva puta kliknete ikonu, prikazaće se informacije o statusu, kao što možete videti na slici 11.1.

SLIKA 11.1 *Informacije o statusu koje prikazuje InterBase kada dva puta kliknete ikonu. License i Capabilities označavaju da je ovo zapravo Local InterBase.*

Postoje dva glavna alata koja možete koristiti da biste direktno komunicirali sa InterBaseom. Jedan je aplikacija Server Manager, koja može da se koristi za administraciju kako lokalnog tako i udaljenog servera, a drugi je Windows Interactive SQL (ili WISQL).

Server Manager može da se koristi za administriranje lokalnih i udaljenih InterBase baza podataka i servera. Upotrebom Server Managera možete manipulisati zaštitom baze podataka (autorizovati nove korisnike, promeniti korisničke lozinke i ukloniti korisničku autorizaciju), načiniti rezervnu kopiju baze podataka, izvršiti zadatke održavanja, prikazati statistiku baze podataka, i izvršiti i druge operacije. Na slici 11.2 je dat primer upotrebe Server Managera za dodavanje novog korisnika.



SLIKA 11.2 *Alat InterBase Server Manager prilikom dodavanja novog korisnika*

Aplikacija Windows InterBase ISQL (Interactive SQL), koja se nalazi u direktorijumu InterBase\Bin, može se koristiti za izvršavanje SQL iskaza na lokalnom i udaljenom InterBase serveru. Možete pokrenuti WISQL, povezati se sa postojećom lokalnom ili udaljenom bazom podataka i uneti SQL iskaz. Na primer, možete se povezati sa alijasom IBLOCAL i uneti sledeći iskaz:

```
select First_Name, Last_Name
from employee
where Job_Code = "Eng"
```

Ova SQL komanda daje ime i prezime svakog zaposlenog iz odeljenja projektovanja (Engineering), kao što možete videti na slici 11.3. Windows ISQL se može koristiti za prikazivanje sadržaja baze podataka, ali je njegova glavna uloga u podešavanju baze podataka i njenom održavanju. Možete definisati nove tabele, dodati indekse, napisati uskladištene procedure i tako dalje. Sve ove operacije možete obaviti upotrebom dela SQL-a Data Definition (definicija podataka).

SLIKA 11.3 *Rezultat izvršavanja SQL iskaza u aplikaciji Windows ISQL*

NAPOMENA

Kao alternativu možete upotrebiti generički front end baze podataka da biste prikazali i modifikovali bazu podataka InterBase. Na primer, možete upotrebiti Delphi Database Explorer i Database Desktop za prolazak kroz postojeće tabele ili baze podataka, kao i za umetanje i uklanjanje slogova i izmenu postojećih vrednosti. Na primer, upotrebom Database Explorera možete izvršiti SQL iskaz sličan onom iz WISQL primera. Jednostavnim selektovanjem kartica Data i SQL možete videti sve podatke ili selektovati samo neka polja ili slogove. ■

Alternativa za ove Windows aplikacije namenjene manipulisanju jeste upotreba alata komandne linije koji izvršavaju slične zadatke. Ovi alati (ISQL za upite, GBACK za rezervne kopije, GFIX za popravljanje grešaka i nekoliko drugih) su zgodni za procesiranje u pozadini i kada radite u Unix ili Linux okruženju. (Ipak, još uvek možete koristiti jednostavnije Windows alate sa klijent Windows mašine koja je povezana na Unix server.)

SQL: jezik za definisanje podataka

RDBMS paketi su, uopšte uzev, toliko blisko zasnovani na SQL-u (Structured Query Language strukturni upitni jezik, često se izgovara sekvel — sequel), da se često nazivaju SQL serverima. SQL je definisao ANSI/IO komitet, mada mnogi serveri koriste specifična proširenja poslednjeg zvaničnog standarda (nazvanog SQL-92 ili SQL2). U poslednje vreme, mnogi serveri su počeli da dodaju objektna proširenja koja bi trebalo da budu deo budućeg standarda SQL3.

Nasuprot onome što ime treba da sugeriše, SQL se koristi ne samo za postavljanje upita nad bazom podataka i manipulaciju podacima, već i za njihovo definisanje. SQL se zapravo sastoji iz dva dela. To su Data Definition Language (DDL — jezik za definisanje podataka) i Data Manipulation Language (DML — jezik za manipulisanje podacima), uključujući i komande upita.

DDL komande se koriste samo prilikom struktuiranja i održavanja baze podataka; njih klijent aplikacija ne koristi direktno. Polazna tačka je komanda create database koja ima jednostavnu sintaksu:

```
create database"mddb.gdb";
```

Ova komanda kreira novu bazu podataka (novi GDB fajl) u trenutnom direktorijumu ili na naznačenoj putanji. Upotrebom WISQL-a takođe možete kreirati bazu podataka upotrebom komande menija File➥Create Database. Primetite tačku i zarez (;) u prethodnom iskazu koji se u WISQL-u koristi kao terminator komande. Suprotna operacija je `drop database`, a takođe možete izmeniti i neke parametre kreiranja upotrebom `alter database`.

NAPOMENA

Klijent programi ne bi trebalo da operišu sa metapodacima, operacijom koja bi se u većini organizacija takmičila sa odgovornostima administratora baze podataka. Ja sam dodao ove pozive jednostavnom Delphi programu (nazvanom DdlSample) samo da bih Vam omogućio da kreirate nove tabele, indekse i okidače u primeru baze podataka. Ovaj primer možete koristiti prilikom čitanja narednog odeljka. Alternativno, ove komande možete uneti u aplikaciju Windows Interactice SQL. ■

Tipovi podataka

Posle kreiranje baze podataka, možete početi da dodajete tabele komandom `create table`. Prilikom kreiranja tabela morate navesti tip podataka za svako polje. SQL sadrži veliki broj tipova podataka, mada je to manji broj nego u Paradoxu i drugim lokalnim bazama podataka. U tabeli 11.1 su dati standardni SQL tipovi podataka i neki drugi tipovi podataka koji postoje na većini servera.

Tabela 11.1: Tipovi podataka koji se koriste u SQL-u

Tip podataka	Standard	Upotreba
`char, character (n)`	Da	Označava string od n karaktera. Određeni serveri ili drajveri imaju ograničenje veličine (32 767 karaktera kada je u pitanju InterBase).
`int, integer`	Da	Celi brojevi, obično četiri bajta, ali to zavisi od platforme.
`smallint`	Da	Manji celi brojevi, obično dva bajta.
`float`	Da	Broj u pokretnom zarezu.
`double precision`	Da	Broj u pokretnom zarezu, velike preciznosti.
`numeric (precision, scale)`	Da	Broj u pokretnom zarezu, sa naznačenom preciznošću i merom.
`date`	Ne	Datum. Implementacija ovog tipa podataka se razlikuje od servera do servera.
`blob`	Ne	Objekat koji sadrži veliku količinu binarnih podataka (BLOB je skraćenica od Binary Large Object — veliki binarni objekat).
`varchar`	Ne	String promenljive dužine koji se koristi da bi se izbeglo zauzimanje memorije kakvo je u slučaju kada imamo veliki string fiksne dužine.

NAPOMENA

Klasa `TStringField` u Delphiju može razlikovati tipove podataka `char` i varchar, naznačavajući tip u svojstvu i rešavajući neke probleme koji se javljaju prilikom upotrebe `char` tipa podataka kojem nisu dodati razmaci na kraju stringa u klauzuli `where` iskaza `update`. ■

Programeri koji su navikli na Paradox i druge lokalne mehanizme, verovatno će primetiti odsustvo logičkog ili Boolean tipa podataka, polja za vreme i datum (`date` tip podataka u InterBaseu služi za čuvanje vremena i datuma) i tipa `AUTOINC`, koji daje veoma često upotrebljavan način za dobijanje jedinstvenog ID-a u tabeli. Odsustvo logičkog tipa može stvoriti nekoliko problema prilikom prosleđivanja postojeće aplikacije. Kao alternativu možete koristiti polje `smallint` sa vrednostima 0 i 1 za `True` i `False`, ili možete koristiti domen, što ću objasniti u narednom odeljku. Tip podataka AutoInc je prisutan kod nekih servera, kao što je Microsoft SQL Server, ali ne i kod InterBasea. Ovaj tip podataka se može zameniti upotrebom generatora, što ću kasnije objasniti.

Domeni

Domeni se mogu koristiti za definisanje vrste korisničkog tipa podataka na serveru. Domen se bazira na postojećem tipu podataka, po mogućstvu ograničenom podskupu (kao u Pascal intervalnom tipu). Domen je koristan kao deo definicije baze podataka, jer možete izbeći ponavljanje provere istog opsega nad nekoliko polja, a istovremeno možete definiciju učiniti čitljivijom.

Na primer, ukoliko imate više tabela sa poljem za adresu, možete definisati tip podataka za ovo polje i zatim koristiti taj tip podataka svaki put kada se koristi polje za adresu:

```
create domain AddressType as char (30);
```

Sintaksa iskaza Vam, takođe, omogućava određivanje predefinisane vrednosti i nekih veza, upotrebom iste notacije koja se koristi prilikom kreiranja tabele (što ćemo videti u narednom odeljku). Ovo je kompletna definicija Boolean domena:

```
create domain boolean as smallint default 0
  check (value between 0 and 1);
```

Upotreba i ažuriranje domena (pozivom `alter domain`) čini delimično lakim ažuriranje predefinisanih vrednosti i proverava istovremeno sva polja zasnovana na tom domenu. To je mnogo lakše nego pozivanje komande `alter table` za svaku od tabela koje treba izmeniti.

Kreiranje tabela

U komandi `create table`, posle naziva nove tabele naznačavate definiciju kolona (ili polja) i neke veze tabele. Svaka kolona sadrži tip podataka i neka svojstva:

- `not null` označava da vrednost polja uvek mora biti prisutna (ovaj parametar je obavezan za polja primarnog ključa ili za polja sa jedinstvenom vrednošću, kao što je dalje opisano).
- `default` označava unapred određenu vrednost polja, koja može biti bilo koja od sledećih: data konstantna vrednost, `null` ili `user` (naziv korisnika koji je umetnuo slog).
- Jedna ili više veza, opciono sa nazivom naznačenim ključnom reči `constraint`. Moguće veze su `primary key` (primarni ključ), `unique` (što označava da svaki slog mora imati jednistvenu vrednost polja), `references` (za referisanje na polje druge tabele) i `check` (da bi se naznačila određena provera podataka).

Evo primera koda kojeg možete upotrebiti za kreiranje tabele koja sadrži jednostavne informacije o kupcima:

```
create table customer (
  cust_no integer not null primary key,
  firstname varchar(30) not null,
  lastname varchar(30) not null,
  address varchar(30),
  phone_number varchar(20)
);
```

U ovom primeru smo koristili `not null` za primarni ključ i za polja za ime i prezime koja se ne mogu ostaviti prazna. Veze tabele mogu sadržati primarni ključ koji se sastoji od više polja:

```
create table cutomers (
  cust_no integer not null,
  firstname varchar(30) not null,
  ...
  primary key (cust_no, name)
);
```

NAPOMENA

Najvažnija veza je `constraint reference` koja Vam omogućava da definišete strani ključ (foreign key) za polje. Strani ključ označava da se vrednost polja odnosi na primarni ključ druge tabele (master tabele). Ova zavisnost čini postojanje polja u master tabeli obaveznim. Drugim rečima, Vi ne možete umetnuti slog koji se referiše na nepostojeće polje, niti možete ukloniti polje dok postoje druge tabele koje se referišu na to polje. ■

Kada ste kreirali tabelu, možete je ukloniti komandom drop table, operacijom koja ne mora uspeti ukoliko tabela ima zavisnosti prema drugim tabelama.

Konačno, možete koristiti komandu `alter table` da biste promenili definiciju tabele, uklanjajući ili dodajući jedno ili više polja i veza. Ipak, ne možete promeniti veličinu polja (na primer, `varchar` polje) i da istovremeno zadržite sadržaj tabele. Potrebno je da sadržaj polja kojem menjate veličinu privremeno premestite na drugo mesto, uklonite polje, dodate novo polje istog naziva ali različite veličine, i na kraju vratite podatke u tabelu.

Indeksi

Najvažnija stvar koju treba da imate na umu je ta da indeksi nisu relevantni za strukturu baze podataka i da se ne odnose na matematički relacioni model. Indeks se može smatrati sugestijom RDBMS-u kako da ubrza pristup podacima.

Zapravo, uvek možete izvršiti upit kojim se zahteva određeno sortiranje, koje se može dobiti nezavisno od indeksa (mada RDBMS može generisati privremeni indeks). Naravno, definisanje i održavanje velikog broja indeksa može zahtevati dosta vremena; ukoliko ne znate tačno kako će to uticati na server, jednostavno prepustite RDBMS-u da kreira indekse koji su mu potrebni.

Kreiranje indeksa se zasniva na komandi `create index`:

```
create index cust_name on customers (name);
```

Kasnije možete ukloniti indeks pozivom drop `index`. InterBase takođe omogućava upotrebu komande `alter index` kojom se indeks može privremeno onemogućiti (parametrom `inactive`) i ponovo staviti u upotrebu (parametrom `active`).

Pogledi

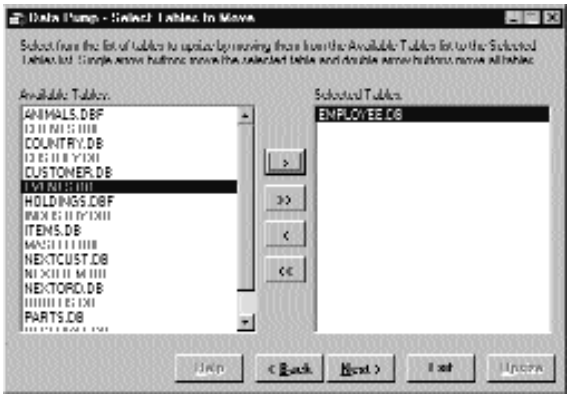
Pored kreiranja tabela, baza podataka Vam omogućava da kreirate poglede nad tabelom. Pogled se definiše iskazom `select` i omogućava Vam da kreirate stalne *virtuelne* tabele koje su mapirane nad fizičkim. U Delphiju pogledi izgledaju identično tabelama.

Pogledi su zgodni za pristupanje spajanjima više puta, ali Vam takođe omogućavaju da ograničite određenim korisnicima prikazivanje podataka (da zabranite pristup osetljivim podacima). Kada je iskaz `select` koji definiše pogled jednostavan, pogled se takođe može ažurirati, zapravo ažurirajući fizičke tabele nad kojima je napravljen; u suprotnom, ukoliko je iskaz `select` složen, pogled će biti samo za čitanje.

PREMEŠTANJE POSTOJEĆIH PODATAKA

Postoje dva alternativna načina definisanja baze podataka nasuprot ručnom pisanju DDL iskaza. Jedna mogućnost je upotreba CASE alata za dizajniranje baze podataka i prepuštanje alatu da generiše DDL kod. Druga je prebacivanje postojeće baze podataka sa jedne platforme na drugu, moguće iz lokalne baze podataka na SQL server. Enterprise verzija Delphija sadrži alat kojim se ovaj proces automatizuje, a to je alat Data Pump Wizard.

Namena ovog alata je izdvajanje strukture baze podataka i njeno ponovno kreiranje na drugoj platformi. Pre nego što pokernete Data Pump, potrebno je da upotrebite BDE Administrator za kreiranje alijasa za bazu podataka koju želite da kreirate. Upotreba Data Pumpa je prilično jednostavna. Potrebno je da selektujete alijas izvora i alijas cilja (odredišta); zatim selektujete tabele koje želite da premestite.



Kada selektujete tabelu (na primer, tabelu EMPLOYEE) i kliknete Next, Data Pump Wizard će verifikovati da li je operacija premeštanja moguća. Posle nekoliko sekundi čarobnjak će prikazati listu tabela i omogućiće Vam da promenite nekoliko opcija.

Ukoliko konverzija polja nije direktna, Data Pump će prikazati poruku Has Problems (postoje problemi) ili Modified (izmenjeno). Posle opcija modifikacije, ukoliko je potrebno, možete kliknuti kontrolu Upsize da biste ste izvršili konverziju. Selektovanjem polja možete verifikovati kako čarobnjak planira da ga prevede; zatim, ukoliko kliknete kontrolu Modify Table Name ili kontrolu Field Mapping Information For Selected Item, možete promeniti postojeću definiciju.

Alternativa za upotrebu Data Pump Wizarda (koji Vam je na raspolaganju samo u verziji Enterprise) je komponenta BatchMove, koja izvršava unapred određenu konverziju tabela i ne može se podešavati. Konačno, možete da upotrebite Database Desktop, kreirate nove tabele na serveru i kliknete kontrolu Borrow Struct da biste izdvojili definiciju tabele iz postojeće lokalne tabele.

SQL: Jezik za manipulaciju podacima

SQL komande iz jezika za manipulaciju podacima (Data Manipulation Language) programeri često koriste, pa ću ih ja detaljnije objasniti. Postoje četiri glavne komande: `select`, `insert`, `update` i `delete`. Sve ove komande se mogu aktivirati upotrebom komponente Query, ali samo `select` daje rezultujući skup podataka. Za sve ostale komande potrebno je da otvorite upit upotrebom metoda `ExecSQL` umesto metoda `Open` (ili svojstva `Active`).

Select

Iskaz `select` je najčešća i najpoznatija SQL komanda; koristi se za izdvajanje podataka iz jedne ili više tabela (ili pogleda) baze podataka. U svojoj najjednostavnijoj formi, komanda izgleda ovako:

```
select <fiedls> from <table>
```

U delu `<fields>` možete navesti jedno ili više polja tabele, odvajajući ih zarezima, upotrebiti simbol * da biste odjednom naznačili sva polja tabele, ili čak naznačiti operaciju koja se izvršava nad jednim ili više polja. Evo jednog složenijeg primera:

```
select upper(name), (lastname || "," || firstname) as fullname
from customers
```

U ovom kodu upper je funkcija servera koja konvertuje sve karaktere u velika slova, dvostruka uspravna crta (||) je operator spajanja stringova, a opciona ključna reč dodeljuje novi naziv celom izrazu koji uključuje ime i prezime.

Dodavanjem klauzule `where` možete upotrebiti iskaz `select` da biste naznačili koje slogove treba izdvojiti kao i polja za koja ste zainteresovani:

```
select *
from customers
where cust_no = 100
```

Ova komanda selektuje jedan slog, onaj koji odgovara kupcu čiji je ID broj jednak 100. Za klauzulom `where` sledi jedan ili više kriterijuma selektovanja, koji mogu biti spojeni upotrebom operatora `and`, `or` i `not`. Evo i primera:

```
select *
from customers
where cust_no = 100 or cust_no = 200
```

Kriterijum selekcije može da sadrži funkcije koje su dostupne na serveru i mogu se koristiti standardni operatori, uključujući +, -, >, <, =, <>, >= i <=. Postoji i nekoliko specijalnih SQL operatora:

- `is null` testira da li je vrednost polja definisana;
- `in <list>` daje vrednost True ukoliko se vrednost nalazi u listi koja sledi za operatorom;
- `between <min> and <max>` proverava da li je vrednost u naznačenom opsegu.

Evo primera ovih operatora:

```
select *
from customers
where address is not null and cust_no between 100 and 150
```

Još jedan moćan operator koji se koristi za proveru dela stringa jeste operator `like`. Na primer, ukoliko želite da pronađete sva imena koja počinju slovom B, možete upotrebiti ovaj iskaz:

```
select *
from employee
where last_name like "B%"
```

Simbol % predstavlja bilo koju kombinaciju karaktera, a može se koristiti i u okviru stringa. Na primer, sledeći iskaz pronalazi sva imena koja počinju slovom B, a završavaju se slovom n:

```
select *
from employee
where upper(last_name) like "B%N"
```

Upotrebom funkcije upper pretraživanje obraća pažnju na velika i mala slova, a to je potrebno jer operator `like` upoređuje stringove pazeći na velika i mala slova. Alternativa za upotrebu operatora `like` je upotreba operatora `containing` i `starting with`. Upotreba operatora `like` nad indeksiranim poljem u InterBaseu može dati veoma sporo pretraživanje jer server neće uvek upotrebiti indeks. Ukoliko tražite upoređivanje sa početnim delom stringa, bolje je koristiti izraz `starting with` koji koristi indeks i mnogo je brži.

Još jedna opcija je sortiranje informacija koje daje iskaz `select` navođenjem klauzule `order by`, u kojoj možete koristiti jedno ili više selektovanih polja:

```
select *
from employee
order by lastname
```

Operatori `asc` i `desc` se mogu koristiti za sortiranje u rastućem ili opadajućem redosledu. Unapred određeno sortiranje je u rastućem redosledu.

Važna varijacija komande `select` je kada koristite klauzulu `distinct` kojom se iz rezultujućeg skupa podataka uklanjaju elementi koji se ponavljaju. Na primer, sledećim iskazom možete prikazati sve gradove u kojima se nalaze Vaši kupci:

```
select distinct city
from customer
```

Komanda `select` se takođe može koristiti za izdvajanje agregatnih vrednosti, koje se izračunavaju standardnim funkcijama:

- `avg` izračunava srednju vrednost kolone rezultujućeg skupa podataka (može se koristiti samo nad numeričkim poljima);
- `count` izračunava broj elemenata rezultujućeg skupa podataka, to jest, broj elemenata koji zadovoljava dati uslov;
- `max` i `min` izračunavaju najveću i najmanju vrednost kolone u rezultujućem skupu podataka;
- `sum` izračunava ukupan zbir vrednosti kolone rezultujućeg skupa podataka (može se koristiti samo nad numeričkim poljima).

Ove funkcije se obavljaju nad rezultujućim skupom podataka, obično nad kolonom, i ne uključuju `null` vrednosti. Sledeći iskaz izračunava prosečnu platu:

```
select avg(salary)
from employee
```

Još jedna važna klauzula je `group by`, koja Vam omogućava da grupišete elemente rezultujućeg skupa podataka prema kriterijumu pre nego što se izračunaju agregatne vrednosti funkcijama koje smo malopre naveli. Na primer, možda želite da odredite maksimalnu i prosečnu platu zaposlenih u svakom od odeljenja:

```
select max(salary), avg(salary), department
from employee
group by department
```

Zapamtite da se sva polja koja se ne izračunavaju, moraju navesti u klauzuli `group by`. Sledeći iskaz nije ispravan:

```
select max(salary), lastname, department
from employee
group by department
```

SAVET

Kada izdvajate agregatne vrednosti, bolje je da koristite alijas za rezultujuće polje upotrebom ključne reči `as`. To olakšava referisanje na rezultujuću vrednost u Vašem Delphi kodu. ■

Agregatne vrednosti se, takođe, mogu koristiti za određivanje slogova u rezultujućem skupu podataka. Agregatne funkcije se ne mogu koristiti u klauzuli `where`, ali se smeštaju u specifičnu klauzulu `having`. Naredni iskaz prikazuje najveću platu u svakom od odeljenja, ali samo ukoliko je vrednost veća od 40000:

```
select max(salary) as maxsal, department
from employee
group by department,
having max(salary) > 40000
```

Još jedna zanimljiva mogućnost je ugnežđenje iskaza `select` unutar drugog iskaza select, čime se formira podupit. Evo primera kojim se prikazuje zaposleni ili više njih koji imaju najveću platu:

```
select firstname, lastname
from employee
where salary = (select max(salary) from employee)
```

Ovaj kod nismo mogli da napišemo upotrebom samo jednog iskaza, jer bi dodavanje imena u rezultat upita impliciralo upotrebu klauzule `group by`.

Unutrašnja i spoljašnja spajanja

Do sada su naši primeri iskaza `select` radili sa jednom tabelom, ozbiljnim ograničenjem relacione baze podataka. Operacija spajanja podataka iz više izvornih tabela se naziva spajanje tabela (table join). SQL standard podržava dva tipa spajanja koja se nazivaju unutrašnje (inner) i spoljašnje (outer).

Unutrašnje spajanje se može napisati direktno unutar klauzule `where`:

```
select *
from <table1>, <table2>
where <table1.keyfield>=<table2.keyfield>
```

Ovo je tipičan primer unutrašnjeg spajanja koje se koristi za izdvajanje svih polja tabela koje se koriste u upitu. Unutrašnje spajanje je pogodno za tabele sa zavisnošću jedan prema jedan (jedan slog prve tabele odgovara samo jednom slogu druge tabele). Zapravo, trebalo bi da standardna sintaksa bude sledeća, mada oba pristupa daju isti efekat:

```
select *
from <table1> left join <table2>
on <<table1.keyfield>=<table2.keyfield>
```

Spoljašnje spajanje se, umesto toga, može precizno zahtevati sledećim iskazom:

```
select *
from <table1> left outer join <table2>
where <<table1.keyfield>=<table2.keyfield>
```

Osnovna razlika između unutrašnjeg i spoljašnjeg spajanja je u tome da selektovani redovi spoljašenjg spajanja neće uzeti u obzir null vrednosti druge tabele. Postoje i drugi tipovi spoljašnjih spajanja, uključujući sledeće: spajanje tabele sa samom sobom (self-join), višestruko spajanje (multi-join) kada je uključeno više od jedne tabele i Dekartov proizvod, spajanje bez klauzule `WHERE`, koje spaja svaki red tabele sa svakim redom druge tabele i obično proizvodi ogroman skup podataka. Unutrašnje spajanje je svakako najčešći primer spajanja.

Insert

Komanda `insert` se koristi za dodavanje novih redova tabeli ili pogleda koji može da se ažurira. Kada umećete novi slog u DBGrid koji je povezan sa tabelom na SQL serveru, BDE generiše komandu insert i šalje je serveru. Pored ove implicitne upotrebe postoji nekoliko slučajeva u kojima ćete želeti da napišete eksplicitne SQL `insert` pozive (uključujući upotrebu keširanih ažuriranja, koja ćemo razmatrati kasnije u ovom poglavlju).

Izuzev ukoliko ne dodajete vrednost svakom polju, potrebno je da navedete nazive polja koja obezbeđujete, kao što se to može videti iz sledećeg koda:

```
insert into employee (empno, lastname, firstname, salary)
values (0, 'brown', 'john', 10000)
```

U tabelu, takođe, sledećom sintaksom, možete umetnuti rezultujući skup podataka iskaza select (ukoliko polja odredišne tabele imaju istu strukturu kao selektovana polja):

```
insert into <table> <select iskaz>
```

Update

Komanda `update` modifikuje jedno ili više polja tabele ili pogleda. Delphi generiše poziv update svaki put kada Vi menjate podatke upotrebom vizuelnih komponenata povezanih sa tabelom ili upitom na SQL serveru. Još jednom, postoje slučajevi kada ćete želeti da direktno koristite iskaz `update`.

Iskazom `update` možete naznačiti koji slog treba izmeniti, upotrebom uslova `where` sličnog onome iz iskaza `select`. Na primer, sledećim pozivom možete promeniti platu određenog zaposlenog:

```
update employee
set salary = 30000
where emp_id = 100
```

UPOZORENJE

Jedna `update` konstrukcija može ažurirati sve slogove koji zadovoljavaju dati uslov. Nekorektna klauzula `where` može slučajno da ažurira mnogo slogova i prikazaće se poruka o grešci. ■

Klauzula `set` se može odnostiti na više polja, odvojenih zarezom, i može koristiti trenutne vrednosti polja za izračunavanje novih vrednosti. Na primer, sledeći iskaz daje lepu povišicu zaposlenima koji su počeli da rade pre 1. januara 1990. godine:

```
update employee
set salary = salary * 10
where hiredate < "01-01-1990"
```

Delete

Komanda `delete` je jednako jednostavna (mada njena pogrešna upotreba može biti prilično opasna). Ponovimo, Vi ćete obično uklanjati slogove upotrebom vizuelnih komponenata, ali takođe možete zahtevati SQL komandu kakva je sledeća:

```
delete from employee
where emp_id = 120
```

Potrebno je da samo navedete uslov kojim se identifikuju slogovi koje treba ukloniti. Ukoliko ovu SQL komandu zahtevate upotrebom komponente Query (pozivom `ExecSQL`), možete upotrebiti svojstvo `RowsAffected` da biste videli koliko je slogova uklonjeno. Isto važi i za komande update.

Upotreba SQL Buildera

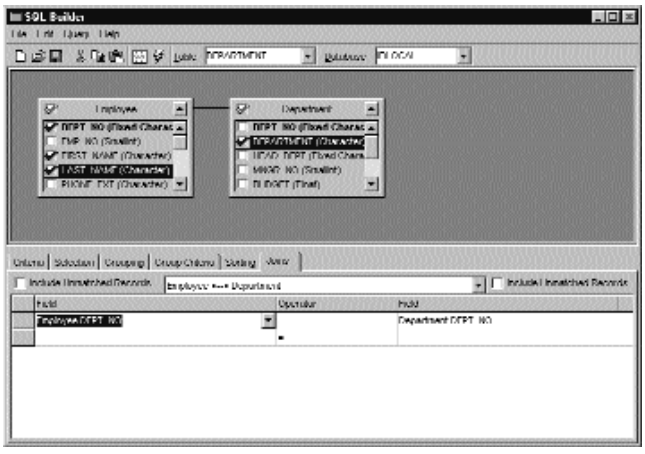
Kao što smo videli, SQL sadrži veliki broj komandi, naročito kada su u pitanju `select` iskazi, a zapravo ih sve nismo videli. Dok DDL komande uobičajeno koriste administratori, ili se koriste samo za inicijalno struktuiranje baze podataka, DML komande se često koriste u svakodnevnom programiranju u Delphiju.

Da bi se pomoglo korektno pisanje SQL iskaza, Delphi Enterprise sadrži alat koji je nazvan SQL Builder. Ovaj alat se lako može aktivirati ukoliko desnim tasterom miša klikente komponentu Query.

U osnovi dodat Delphiju 4 da bi se zamenio više nego ograničeni Visual Query Builder, koji se može naći u ranijim verzijama, SQL Builder je dvosmerni alat; možete ga koristiti kako za kreiranje tekstualnih upita tako i za dobijanje grafičkog prikaza postojećeg upita (čak i ukoliko ste izmenili originalni tekst).

Upotreba SQL Buildera je veoma jednostavna. Vi birate bazu podataka sa kojom želite da radite, a zatim selektujete jednu ili više tabela smeštajući ih na radnu površinu. Posle selektovanja odgovarajućih parametara, što ću objasniti, možete upotrebiti Query➥Run Query (ili pritisnuti taster F9) da biste prikazali rezultat upita, ili Query➥Show SQL (ili pritisnuti taster F7) da biste prikazali izvorni kod `select` iskaza koji ste generisali.

U selektovanim tabelama možete jednostavno označiti polja koja želite da se prikažu u rezultujućem skupu podataka. Polje za potvrdu pored naziva tabele potvrđuje sva polja tabele. Međutim, prava snaga SQL Buildera se nalazi u dvema novim funkcijama. Prvo, možete prevući polje iz jedne tabele u drugu da biste spojili tabele, kao što je prikazano na slici 11.4.



SLIKA 11.4 *Dve tabele koje su prikazane kao spojene u SQL Builderu*

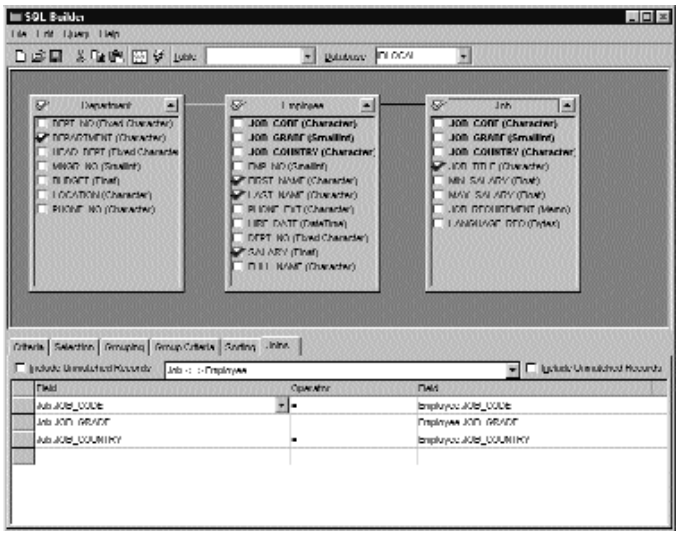
Druga moćna funkcija je Query beležnica, kontrola sa više strana koja se nalazi na dnu prozora SQL Builder. Evo kratkog opisa svake od strana:

- Strana *Criteria* prikazuje kriterijum selekcije klauzule `where`. Selektovanjem jednog od polja rezultujuće tabele, možete naznačiti poređenje između fiksne vrednosti i drugog polja, i možete upotrebiti `like, is null, between` i druge operatore. Upotrebom lokalnog menija tabele, koji se nalazi na ovoj strani, takođe možete aktivirati operator `exist` ili ceo SQL iskaz. Ova strana Vam omogućava da kombinujete više uslova upotrebom operatora `and`, `or` i `not`, ali Vam ne omogućava da naznačite prioritet operatora upotrebom zagrada.
- Strana *Selection* prikazuje sva polja rezultujućeg skupa podataka i omogućava Vam da svima dodelite alijas. Upotrebom lokalnog menija takođe možete uvesti agregatne funkcije (`sum, count` i tako dalje). Konačno, gornje levo polje za potvrdu označava uslov `distinct`.
- Strana *Grouping* odgovara klauzuli `group by`. SQL Builder automatski grupiše sva polja koja se koriste u agregatnim funkcijama, kao što zahteva SQL standard.
- Strana *Group Criteria* odgovara klauzuli `having`, koja je na raspolaganju uz agregatne funkcije. Operacije su slične operacijama sa strane Selection i aktiviraju se upotrebom lokalnog menija.
- Strana *Sorting* odgovara klauzuli `order by`. Jednostavno odaberete polje koje želite da sortirate, a zatim odaberete rastući ili opadajući redosled.
- Strana *Joins* je poslednja, ali verovatno najmoćnija strana, jer Vam omogućava da definišete uslove spajanja, više od jednostavnog prevlačenja polja iz jedne do druge tabele na radnoj površini. Ova strana Vam omogućava da fino podesite spajanje koje se zahteva, naznačavanjem tipa spajanja (`INNER` ili `OUTER`) i odabiranjem uslova koji nisu poređani po jednakosti.

Da biste bolje razumeli kako da koristite SQL Builder, mi ćemo izraditi primer nad datom InterBase bazom podataka koju je instalirao Delphi (i koja odgovara alijasu LocalIB). Primer se nalazi u direktorijumu SqlBuilder i njegov formular sadrži komponente Query, DataSource i DBGrid koje su povezane na uobičajen način. Svojstvo `DatabaseName` komponente Query je podešeno na IBLocal, a kada komponentu kliknete desnim tasterom miša, aktiviraćete SQL Builder, kao što je prikazano na slici 11.5.

Mi želimo da kreiramo upit koji uključuje ime i prezime, odeljenje, zvanje i platu svakog zaposlenog. Ova operacija zahteva dva spajanja. Odaberite tabele Employee, Department i Job. Kliknite polje Dep_No tabele Department i prevucite kursor nad polje Dep_No tabele Employee. Slično tome, povežite tabelu Job sa tabelom Employee koristeći tri polja: Job_Code, Job_Grade i Job_Country.

Posle kreiranja spajanja odaberite polja koja želite da prikažete u rezultujućem skupu podataka: First_Name, Last_Name i Salary iz tabele Employee; Department iz tabele Department; Job_Title iz tabele Job. Konačno, pređite na stranu Sorting beležnice Query i odaberite Department.Department iz liste Output Fields da biste sortirali rezultat prema odeljenjima.



SLIKA 11.5 *Složeno spajanje prikazano u SQL Builderu*

Ono što sledi je SQL koji bi trebalo da bude generisan:

```
select employee.first_name, employee.last_name,
  department.department, job.job_title, employee.salary
from employee employee
  inner join department department
  on (department.dept_no = employee.dept_no)
  inner join job job
  on (job.job_code = employee.job_code)
  and (job.job_grade = employee.job_grade)
  and (job.job_country = employee.job_country)
order by department.department
```

Možemo da dodamo klauzulu `where` da bismo odabrali samo zaposlene sa velikom platom. Jednostavno pređite na stranu Selection, odaberite polje Employee.Salary, pređite na kolonu Operator >= i unesite **100,000**. Izvršavanjem upita prikazaće se samo ograničen broj slogova, a kada pogledate izvorni kod, videćete dodatni iskaz:

```
where employee.salary >= 100000
```

Konačno, zapamtite da je moguće izvesti i uvesti SQL kod iz običnog tekst fajla. Jednostavno, zatvaranjem SQL Buildera ćete sačuvati tekst upita u SQL svojstvu odgovarjuće komponente Query.

SAVET

Kada radite sa ADO upitima ili InterBase Express upitima, nećete na raspolaganju imati snagu SQL Buildera. Možete, ipak, pripremiti upit SQL Builderom koristeći generičku komponentu `TQuery` i zatim kopirati SQL kod u komponentu Query koju želite da koristite. ■

Programiranje servera

Na početku ovog poglavlja sam istakao činjenicu da je jedan od ciljeva klijent/server programiranja — i jedan od problema — podela posla između kompjutera koji su uključeni u posao. Kada aktivirate SQL iskaze sa klijenta, teret većeg dela posla pada na server. Ipak, trebalo bi da pokušate da koristite `select` iskaze koji daju veliki rezultujući skup podataka, da bi se izbeglo zagušenje mreže.

Pored prihvatanja DDL i DML zahteva, većina RDBMS servera Vam omogućava da rutine kreirate direktno na serveru koristeći standardne SQL komande kao i njihova specifična proširenja, koja zavise od servera (koja, uopšte uzev, nisu prenosiva). Ove rutine tipično postoje u dve forme: kao uskladištene procedure i okidači.

Uskladištene procedure

Uskladištene procedure su nešto nalik na globalne funkcije Delphi jedinica i moraju se eksplicitno pozvati od strane klijenta. Uskladištene procedure se obično koriste za definisanje rutina za održavanje podataka, za grupisanje niza operacija koje su Vam potrebne za različite situacije, ili za čuvanje složenih `select` iskaza.

Kao i Pascal procedure, uskladištene procedure mogu imati jedan ili više parametara koji se unose i mogu kao rezultat dati neku vrednost. Kao alternativu za vraćanje vrednosti, uskladištena procedura takođe može dati rezultujući skup podataka, rezultat internog `select` iskaza.

Ono što sledi je uskladištena procedura napisana za InterBase, za koju treba uneti datum, a koja izračunava najveću platu koju imaju zaposleni koji su u radni odnos stupili tog datuma:

```
create procedure maxsaloftheday (ofday date)
returns (maxsal decimal(8, 2))
as
begin
  select max(salary)
  from employee
  where hiredate = :ofday
  into :maxsal;
end
```

Primetićete upotrebu klauzule into koja serveru govori da sačuva rezultat `select` iskaza u vrednosti `MaxSal` koju daje kao rezultat. Da biste izmenili ili uklonili uskladištenu proceduru, kasnije možete upotrebiti komande `alter procedure` i `drop procedure`.

Kada pogledate ovu uskladištenu proceduru, možda se pitate kakve su njene prednosti nad sličnim upitom koji se aktivira sa klijenta. Razlika između dva pristupa nije u rezultujućem skupu podataka koji se dobija, već u brzini. Uskladištena procedura se kompajlira na serveru u prelaznoj i bržoj notaciji, a server će u tom trenutku odabrati strategiju koju će koristiti za pristupanje podacima. Nasuprot tome, upit se kompajlira svaki put kada se zahtev šalje serveru (mada server može keširati upit i time izbeći ponovno kompajliranje dva identična zahteva). Zbog toga uskladištena procedura može zameniti veoma složen upit, pod uslovom da se ne menja često.

Iz Delphija možete aktivirati uskladištenu proceduru koja daje rezultujući skup upotrebom bilo komponente Query bilo Stored Procedure. Upotrebom komponente Query možete napisati sledeći kod:

```
select *
from MaxSalOfTheDay ("01/01/1990")
```

Uglavnom je lakše koristiti StoredProc kada procedura ima više parametara, ili kada ima složene parametre. Ova komponenta prikazuje uskladištene procedure koje su na raspolaganju na serveru, i sadrži okvir za dijalog koji je lak za upotrebu i koji može poslužiti za definisanje parametara, kao što je prikazano na slici 11.6.

SLIKA 11.6 *Editor svojstva Params komponente StoredProc*

Okidači (i generatori)

Okidači (trigger) se više ili manje ponašaju kao Delphi događaji i automatski se aktiviraju kada se desi dati događaj. Okidači mogu imati specifični kod ili mogu pozivati uskladištene procedure. U oba slučaja izvršavanje se potpuno obavlja na serveru. Okidači se koriste da bi podaci ostali konzistentni, proverom podataka na mnogo složeniji način nego što je provera kada postoje zavisnosti, i za automatizovanje sporednih efekata nekih operacija unosa (kao što su kreiranje dnevnika prethodnih izmena plata).

Okidači se mogu aktivirati pomoću tri osnovne operacije ažuriranja podataka: `insert`, `update` i `delete`. Kada kreirate okidač, Vi naznačavate da li se izvršava pre ili posle jedne od ove tri akcije.

Kao primer okidača možemo upotrebiti generator ili niz za kreiranje jedinstvenog indeksa nad tabelom. Mnoge tabele koriste jedinstven indeks kao primarni ključ. SQL serveri nemaju polje tipa AutoInc, za razliku od Paradox ili drugih lokalnih baza podataka. Kako više klijenata ne može generisati jedinstvene identifikatore, možemo se osloniti na server da to obavi. Gotovo svi SQL serveri nude brojač koji možete pozvati da biste dobili novi ID, koji ćete kasnije upotrebiti za tabelu. InterBase naziva ove automatske brojače generatorima, dok ih Oracle naziva sekvencama. Evo primera InterBase koda:

```
create generator cust_no_gen;
...
gen_id (cust_no_gen, 1);
```

Funkcija `gen_id` zatim izdvaja novu jedinstvenu vrednost generatora koja se prosleđuje kao prvi parametar, dok drugi parametar označava koliki je korak uvećavanja (u ovom slučaju jedan).

U ovom trenutku okidač možete dodati tabeli, automatskoj obradi jednog od događaja tabele. Okidač je sličan obradi događaja komponente Table, ali Vi SQL kod pišete i izvršavate na serveru, a ne na klijentu. Evo i primera:

```
create trigger set_cust_no for customers
before insert position 0 as
begin
  new.cust_no = gen_id (cust_no_gen, 1);
end
```

Ovaj okidač je definisan za tabelu Customers i aktivira se svaki put kada se umetne novi slog. Simbol `new` označava novi slog koji umećemo. Opcija `position` označava redosled izvršavanja višestrukih okidača povezanih sa istim događajem. Okidači sa nižim vrednostima će biti prvi izvršeni.

Unutar okidača možete napisati DML iskaze koji, takođe, ažuriraju druge tabele, ali se čuvajte ažuriranja koja ponovo aktiviraju okidač, čime se kreira beskonačna rekurzija. Kasnije možete onemogućiti ili izmeniti okidač pozivom iskaza `drop trigger` ili `alter trigger`.

SAVET

Okidači se automatski aktiviraju za naznačene događaje. Ukoliko je potrebno da načinite mnoge izmene u bazi podataka upotrebom grupnih operatora, postojanje okidača može usporiti proces. Ukoliko su podaci već proverni radi konzistentnosti, možete provremeno ukloniti okidač. Operacije koje se izvršavaju u pozadini, često se kodiraju u uskladištenim procedurama, ali uskladištene procedure uglavnom ne mogu izvršavati DDL iskaze, kao što su iskazi koji su potrebni za uklanjanje i ponovno postavljanje okidača. U takvom slučaju možete definisati pogled jednostavnom komandom `select * from table`, kreirajući tako alijas tabele. Zatim možete prepustiti uskladištenoj proceduri da u pozadini obavi procesiranje nad tabelom i da na pogled primenite okidač (kojeg bi takođe trebalo da koristi klijent program). ■

Aktivni upiti i keširana ažuriranja

Kada radite sa lokalnim podacima, veoma je česta upotreba tabela (grid) i drugih vizuelnih kontrola, često se menjaju podaci i šalju nazad bazi podataka. Do sada smo videli da upotreba komponente DBGrid može prouzrokovati nekoliko problema kada se radi sa RDBMS-om, kao kada se pregleda tabela, jer se serveru šalju brojni zahtevi za podacima, što dovodi do ogromnog mrežnog saobraćaja.

Kada koristite komponentu Query za povezivanje sa nekim podacima, Vi ne možete izmeniti podatke, izuzev ukoliko nije određena vrednost `True` za svojstvo `LiveRequest`. Ukoliko radite sa lokalnim tabelama, upit uvek obrađuje BDE upotrebom mehanizma Local SQL. BDE će dozvoliti aktivni (live) upit samo ukoliko je upit sasvim jednostavan: sva spajanja bi trebalo da budu spoljašnja; ne može se koristiti `distinct`; ne mogu se koristiti agregatne funkcije; ne mogu postojati klauzule `group by` i `having`; ne mogu se koristiti podupiti; ne može se koristiti `sort by`, izuzev ukoliko nije nad indeksom. Postoje još neka pravila koja možete pronaći u Delphijevim Help fajlovima.

Ukoliko radite sa SQL serverom, određivanje aktivnog upita će BDE-u predati odgovornost nad upitom, a ne serveru. Kada ste povezani sa SQL serverom, aktivni upit se ponaša kao komponenta Table. (Dakle, ima smisla koristiti tabelu u ovakvim slučajevima.)

SAVET

Većina SQL servera, uključujući i InterBase, omogućava Vam definisanje pogleda koji se mogu ažurirati, a koji su bazirani na rezultatu iskaza `select` koji Local SQL mehanizam BDE-a neće smatrati pogledima koji se mogu ažurirati. Tada možete povezati komponentu Table sa pogledom, prepuštajući posao SQL serveru i zaobilazeći Local SQL mehanizam BDE-a. ■

Ukoliko BDE odredi da se skup podataka ne može ažurirati, BDE određuje vrednost `False` za svojstvo `CanModify`. Komponenta DataSource proverava ovu vrednost pre nego što omogući operaciju editovanja. Rešenje ovog problema je u izbegavanju upotrebe kontrola koje prepoznaju podatke, što smo razmatrali u prethodnom poglavlju, i u upotrebi specifičnih SQL upita za ažuriranje, umetanje i uklanjanje slogova.

Bolji pristup je automatizovanje ovog procesa (zadržavajući mogućnosti kontrola koje prepoznaju podatke) upotrebom komponente UpdateSQL uz upotrebu komponente Query. UpdateSQL se može koristiti samo uz keširana ažuriranja, temu koju ćemo razmatrati kasnije u ovom poglavlju. Osnovna ideja je da se operacije ažuriranja čuvaju u lokalnom kešu sve dok program ne pozove metod `ApplyUpdates` komponente Query. Ova operacija odgovara izvršavanju niza `update`, `insert` i `delete SQL` operacija na serveru, upotrebom podataka koji se nalaze u kešu. Potrebne SQL komande se čuvaju u komponenti UpdateSQL, koja sadrži editor koji se može koristiti u vreme dizajniranja za gotovo automatsko generisanje ovih SQL komandi.

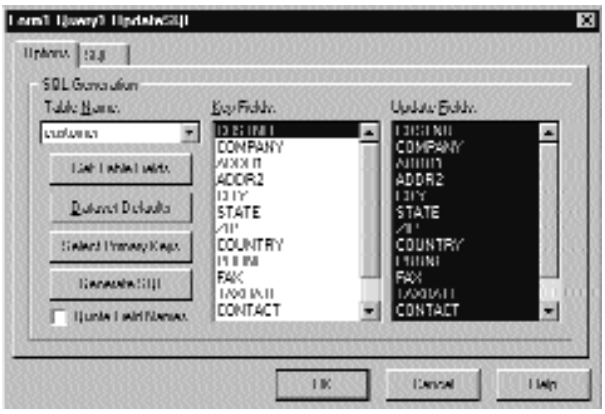
Keširana ažuriranja rešavaju problem aktivnih upita, smanjuju mrežni saobraćaj, definišu standardni način za rešavanje konflikata ažuriranja i smanjuju posao servera, ali zahtevaju više memorije na klijent kompjuteru.

Komponenta UpdateSQL

Uloga komponente UpdateSQL je da obezbedi upit sa iskazima ažuriranja da bi mogli da se ažuriraju rezultati upita. Ključna svojstva komponente su `DeleteSQL`, `InsertSQL` i `ModifySQL`, ali najvažniji element je svojstvo `UpdateObject` odgovarajuće komponente Query. SQL iskazi ažuriranja se izvršavaju kada primenjujete keširana ažuriranja šaljući izmene serveru. Kako keširana ažuriranja zadržavaju informacije o originalnim slogovima, ova ažuriranja obično naznačavaju koji slog treba ažurirati prosleđivanjem originalnih podataka. To je jedini način na koji možemo označiti slog na SQL serveru, a ova tehnika pomaže serveru da prati ažuriranja nad istim slogom koja su načinili drugi korisnici.

Sva ova priprema izgleda kao da implicira dosta posla, ali je zapravo veoma jednostavna. Kada ste napisali upit, Vi povezujete komponentu UpdateSQL sa upitom i aktivirate editor komponete, kao što je prikazano na slici 11.7.

Ovaj editor komponente se saastoji iz dva dela. Na prvoj strani se naznačava kriterijum koji se koristi za generisanje SQL iskaza za dodavanje, uklanjanje ili izmenu slogova. Kada je u pitanju spajanje, možete da selektujete tabelu koju ažurirate i polja koja su uključena u ažuriranje. Kada ste završili ovaj korak, kliknite kontrolu Generate SQL, a editor će preći na drugu stranu gde možete da proverite generisani SQL kod za tri operacije.



SLIKA 11.7 *Editor komponente UpdateSQL prilikom upotrebe*

Primer UpdateSQL

Da bih demonstrirao pravu snagu komponente UpdateSQL, izradio sam primer koji sam nazvao UpdateSQL koji koristi tabele Employee, Department i Job baze podataka IBLocal, koju smo i ranije koristili. Primer se zasniva na upitu koji je prikazan ranije na slici 11.5, i koji je smešten uz druge komponente za pristup podacima u modulu podataka.

Evo tekstualnog opisa komponente UpdateSQL ovog primera:

```
object EmpUpdate: TUpdateSQL
  ModifySQL.Strings = (
     'update EMPLOYEE'
    'set'
    ' FIRST_NAME = :FIRST_NAME,'
    ' LAST_NAME = :LAST_NAME,'
    ' SALARY = :SALARY,'
    ' DEPT_NO = :DEPT_NO,'
    ' JOB_CODE = :JOB_CODE,'
    ' JOB_GRADE = :JOB GRADE,'
    ' JOB_COUNTRY = :JOB_COUNTRY'
    'where'
    ' EMP_NO = :OLD EMP NO')
InsertSQL.Strings = (
    'insert into EMPLOYEE'
    ' (FIRST NAME , LAST NAME , SALARY, DEPT_NO, JOB OOE,
    ' JOB_GRADE, JOB COUNTRY)'
    'values'
    ' (:FIRST NAME, :LAST NAME, :SALARY :DEPT_NO, JOB_CODE,'
    ' :Job_GRADE, :JOB_COUNTRY)')
DeleteSQL.Strings = (
    'delete from EMPLOYEE'
    'where'
    ' EMP_NO = :OLD_EMP_NO')
end
```

Za uklanjanje slogova zaposlenih program koristi uskladištenu proceduru koja je već na raspolaganju u primeru baze podataka i koja je povezana sa sledećom komponentom:

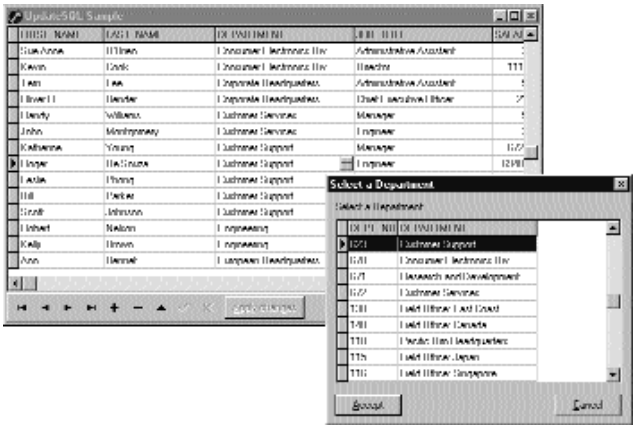
```
object spDelEmployee: TStoredProc
  DatabaseName 'AppDB'
  StoredProcName = 'DELETE_EMPLOYEE'
  ParamData = <
    item
      DataType = ftInteger
      Name = 'EMP_NOSY'
      ParamType = ptInput
    end>
end
```

Događaj `OnUpdateRecord` komponente Query koristi uskladištenu proceduru umesto unapred određene UpdateSQL komponente za uklanjanje slogova. Evo koda za obradu događaja:

```
procedure TdmData.qryEmployeeUpdateRecord(DataSet: TDataSet;
  UpdateKind: TUpdateKind; var UpdateAction: TUpdateAction);
begin
  // when deleting the record, use the stored procedure
  if UpdateKind = ukDelete then
  begin
    // assign empno value
    with dmData do
      spDelEmployee.Params[0].Value:=qryEmployeeEMP_NO.OldValue;
    try
      //invoke stored procedure thar tries to delete employee
      dmData. spDelEmployee.ExecProc;
      UpdateAction := uaApplied; // success
    except
      UpdateAction := uaFail;
    end;
  end
  else
  try
    // apply updates
    dmData.EmpUpdate.Apply(UpdateKind);
    UpdateAction := uaApplied;
  except
    UpdateAction := uaFail;
  end;
end;
```

Primetićete da, zbog toga što operaciju ažuriranja izvodimo direktno, u prametru `UpdateAction` moramo naznačiti da li je operacija uspela ili ne. Ovaj kod je deo modula podataka. Glavni formular, čiji izgled u vreme izvršavanja možete videti na slici 11.8, sadrži i nekoliko drugih funkcija. Ukoliko korisnik zatvori formular kada na izvršavanje čekaju neka ažuriranja, događaj formulara `OnCloseQuery` prikazuje poruku upozorenja omogućavajući korisniku da primeni ažuriranja ili da ih preskoči:

```
procedure TMainForm.FormCloseQuery(Sender: TObject;
  var CanClose: Boolean);
var
  Res: Integer;
begin
  with dmData do
    if qryEmployee.UpdatesPending then
    begin
      Res := MessageDlg (CloseMsg, mtInformation,
        mbYesNoCancel, 0);
      if Res = mrYes then
        AppDB.ApplyUpdates ([qryEmployee]);
      CanClose := Res <> mrCancel;
    end;
end;
```



SLIKA 11.8 *Glavni formular primera UpdateSQL i sekundarni formular istog primera*

Druga karakteristika je upotreba sekundarnog formulara za ažuriranje polja koja se odnose na druge tabele — polja koja su uključena u spajanje. Program koristi dva sekundarna okvira za dijalog koja podatke dobijaju iz Query komponenata. Okviri za dijalog se prikazuju kada korisnik klikne kontrolu sa tri tačke kontrole DBGrid, u događaju `OnEditButtonClick`. Evo prvog dela obrade događaja koja se odnosi na selektovanje odeljenja:

```
procedure TMainForm.DBGrid1EditButtonClick(Sender: TObject);
begin
  // check if this is the department field
  if DBGrid1SelectedField = dmData.qryEmployeeDEPARTMENT then
    with TfrmDepartments.Create(self) do
    try
      dmData.qryDepartment.Locate( 'DEPT_NO',
        dmData.qryEmployeeDEPT_NO.Value []);
      if ShowModal = mrOk then
        with dmData do
        begin
          if not (qryEmployee.State in [dsEdit, dsInsert]) then
```

```
            qryEmployee. Edit;
          qryEmployeeDEPT_NO.Value :=
            qryDepartment.Fields[0].Value;
          qryEmp1oyeeDEPARTMENT.Value :=
            qryDepartment. Fields[1] Value;
        end;
    finally
      Free;
    end
else // similar code for the job fields...
```

Konačno, kontrola Apply jednostavno poziva metod `ApplyUpdates` ukoliko postoje ažuriranja koja čekaju izvršenje, a zatim osvežava podatke upita:

```
procedure TMainForm.btnApplyClick(Sender: TObject);
begin
  with dmData do
    if qryEmployee.UpdatesPending then
    begin
      AppDB.ApplyUpdates( [qryEmployee]);
      // refresh the data
      qryEmployee.Close;
      qryEmployee.Open;
      btnApply.Enabled := False;
    end;
end;
```

Ukoliko pokrenete ovaj program, primetićete da, čak i kada je upit samo za čitanje, možete direktno promeniti podatke u kontroli DBGrid, kao što bi to bio slučaj da je u pitanju bila komponenta Table. Vizuelne operacije koje obavljate privremeno se čuvaju u kešu; zatim, kada zahtevate operaciju ažuriranja, komponente UpdateSQL i StoredProc obezbeđuju kod. Takođe, imajte na umu da polja sa platom sadrže nekakve zavisnosti (definisane u primeru baze podataka), te ih morate pažljivo menjati da biste izbegli greške na serveru kada se izmene primenjuju.

Konflikti ažuriranja

Kada radite sa lokalnim tabelama, upotreba keširanih ažuriranja može dovesti do nekih problema konkurentnosti. Obična operacija editovanja obično zaključava tabelu tako da ostali korisnici ne mogu da izmene isti slog sve dok prvi korisnik ne pošalje izmene. Prethodno poglavlje je razmatralo zaključavanje i konkurentnost.

Kada radite sa SQL serverima, unapred određeno zaključavanje je optimističko. Više korisnika može ažurirati isti slog, ali samo kada se podaci pošalju natrag, server verifikuje originalne podatke sloga pre ažuriranja, a potencijalno može prikazati grešku. Preciznije, iskaz `update` koristi jedno ili više originalnih polja za pronalaženje sloga koji želite da ažurirate. Ukoliko koristite sva polja, a drugi korisnik je promenio slog, server neće pronaći originalni slog i doći će do greške u ažuriranju.

Vi možete ručno kontrolisati ovo ponašanje bilo kodom komponente UpdateSQL (naznačavajući da se uključe sva polja koja su pročitana u upitu), bilo upotrebom svojstva `UpdateMode` komponenata Table i Query. Unapred određena vrednost `upWhereAll` označava da će upit ažuriranja sadržati klauzulu `where` sa svim originalnim poljima sloga. U mnogim slučajevima činjenica da je drugi

korisnik izmenio polje koje nije među poljima koja smo mi menjali, ne dovodi do greške. Možemo odrediti mod `upWhereChanged` da biste prepustili Delphiju da generiše izuzetak i prikaže poruku o grešci, samo ukoliko trenutni korisnik i neki drugi korisnik menjaju ista polja. Treći način je upotreba ključnog polja samo za identifikaciju sloga, što znači da će se konflikti ažuriranja ignorisati i da će poslednji korisnik koji šalje podatke jednostavno izbrisati prethodne izmene. Kao što i pretpostavljate, ovo je opcija koju treba izbegavati u klijent/server višekorisničkom okruženju.

Upotreba transakcija

Još jedna tema koja se odnosi na klijent/server okruženje je upotreba transakcija. U prethodnom poglavlju smo predstavili koncept transakcija (više ažuriranja koja se smatraju kao jedna operacija), ali postoji još detalja koji se odnose na rad sa SQL serverima.

U Delphiju koristimo komponentu Database za rukovanje transakcijama i za određivanje nivoa izolacije transakcije upotrebom svojstva `TransIsolation`. Kada jedan korisnik započne transakciju i izmeni podatke, da li bi trebalo da takve izmene budu vidljive drugim korisnicima? Šta se dešava kada korisnik poništi transakciju? Na takva pitanja ne postoji univerzalan odgovor; svaki programer bi trebalo da na njih odgovori na osnovu zahteva i pravila rada aplikacije. Postoje tri različite vrednosti za nivo izolacije transakcije u BDE-u:

- `tiDirtyRead` odmah čini ostalim transakcijama i korisnicima vidljiva ažuriranja transakcija, čak i kada se ne upišu. Ovo je jedina mogućnost za lokalne baze podataka koje imaju veoma ograničenu podršku transakcija.
- `tiReadCommited` čini drugim transakcijama dostupna samo ažuriranja koja su već upisana.
- `tiRepeatable Read` sakriva sve ostale transakcije koje su započeli drugi korisnici posle započinjanja trenutne transakcije. Naredni pozivi unutar transakcije će uvek dati isti rezultat kao da je baza podataka preuzela zamrznutu sliku podataka kada je započela transakcija.

Većina ali ne i svi SQL serveri podržavaju samo najnaprednije nivoe. Unapred je određen `tiReadCommited` koji je prilično moćan, ali ne pretežak za rad SQL servera (jer postavlja veoma malo internog zaključavanja).

Kao opšta sugestija, transakcije bi trebalo da uključuju samo minimalan broj ažuriranja (samo ona koja su striktno povezana i deo su jedne operacije) i trebalo bi da se obave za kratko vreme. Trebalo bi da izbegavate transakcije koje čekaju na korisnički unos da bi bile kompletne, jer korisnik može privremeno biti odsutan, a transakcija dugo može ostati aktivna. Upotreba iskaza update za više slogova i upotreba keširanih ažuriranja pomaže pri stvaranju malih i brzih transkacija.

Da biste dalje istražili transakcije i eksperimentisali sa modom ažuriranja komponente Table, možete upotrebiti aplikaciju TranSample. Kao što se vidi na slici 11.9, Vi možete da upotrebite dve opcione kontrole za izbor različitih mogućnosti i možete da kliknete kontrole na desnoj strani palete alata da biste ručno pokrenuli, upisali i opozvali transakcije. Da biste dobili pravu sliku o različitim efektima, potrebno je da pokrenete više kopija programa (ukoliko imate dovoljno licenci na Vašem InterBase serveru).

SLIKA 11.9 *Aplikacija TransSample Vam omogućava da testirate izolaciju transakcije baze podataka i modove ažuriranja tabele*

InterBase Express

Svi primeri koje smo do sada izradili, koristili su BDE da bi došli do SQL servera. Kao što sam pomenuo na početku ovog poglavlja, Delphi 5 sadrži novu komponentu naročito dizajniranu za baze podataka InterBase. Ova tehnologija je nazvana InterBase Express (ili skraćeno IBX). Borland obećava da će aplikacije koje koriste ove komponente raditi bolje i brže, i da daju veću kontrolu nad specifičnim krakteristikama InterBasea. Ja nemam razloge da u to sumnjam. Problem je u tome što aplikacija izrađena ovim pristupom nije prenosiva na druge SQL servere.

IBX komponente sadrže korisnički skup komponenata za baze podataka. Ove su izvedene iz klase `TDataSet`, mogu koristiti sve Delphi kontrole koje prepoznaju podatke, obezbeđuju editore polja i sve uobičajene pogodnosti u vreme izvršavanja, i mogu se koristiti u novom Data Module Designeru, ali ne zahtevaju BDE. Vi, zapravo, možete da birate između više komponenata skupa podataka:

- IBTable podseća na komponentu Table i omogućava pristup jednoj tabeli ili pogledu.
- IBQuery podseća na komponentu Query i omogućava izvršavanje SQL upita dajući rezultujući skup podataka. Komponenta IBQuery se može koristiti uz komponentu IBUpdateSQL da bi se dobio aktivni (ili izmenljivi) skup podataka.
- IBStoredProc podseća na komponentu StoredProc i omogućava izvršavanje uskladištene procedure.
- IBDataSet omogućava rad sa aktivnim rezultujućim skupom podataka dobijenim izvršavanjem upita `select`. Ova komponenta u osnovi spaja IBQuery sa IBUpdateSQL u jednu komponentu.

Mnoge druge komponente u InterBase Expressu ne pripadaju kategoriji skupa podataka:

- IBDatabase oponaša standardnu komponentu Database.

- IBTransaction omogućava potpunu kontrolu nad transakcijama.
- IBSQL omogućava izvršavanje SQL iskaza, a da ne utiče na kontrolu skupa podataka.
- IBDatabaseInfo se koristi za ispitivanje strukture baze podataka i statusa baze podataka.
- IBSQLMonitor se koristi za debagovnje sistema.
- IBEvents prihvata događaje koje šalje server.

Ova velika grupa kontrola obezbeđuje veću kontrolu nad serverom baze podataka nego što je ima BDE. Na primer, pošto postoji komponenta za transakcije, to nam omogućava upravljanje većim brojem konkurentnih transakcija nad jednom ili više baza podatka, kao i kada se jedna transakcija odnosi na više baza podataka.

Spreman i izvršava se

Kao prvi primer ja sam odabrao program IbEmpl koji smo razmatrali ranije, i ponovio sam ga minimalno koristeći potrebne InterBase komponente. Kada sam komponentu Query zamenio komponentom IBQuery, morao sam da dodam još dve komponente: IBTransaction i IBDatabase. Bilo koja IBX aplikacija zahteva bar po jednu instancu ovih dveju komponenata. Ne možete podesiti vezu baze podataka sa skupom podataka (kao kada je to slučaj sa komponentom Query), i potreban je još bar objekat za transakciju da bi se otvorio upit.

Evo ključnih svojstava ovih komponenata u primeru IbEmpl2:

```
object IBTransaction1: TIBTtransaction
  Active = False
  DefaultDatabase = IBDatabase1
end
object IBQuery1: TIBQuery
  Database = IBDatabase1
  Transaction = IBTransaction1
  CacheUpdates = False
  SQL.Strings = (
    'SELECT * FROM EMPLOYEE')
end
object IBDatabase1: TIBDatabase
  DatabaseName = 'C:\Program Files\Common Files\
    Borland Shared\Data\employee.gdb'
  Params.Strings = (
    'user_name = SYSDBA'
    'password=masterkey')
  LoginPrompt = False
  IdleTimer = 0
  SQLDialect = 1
  TraceFlags = []
end
```

Da biste izvršili izmene, nije potrebno mnogo vremena, i ukoliko pristupate istoj tabeli baze podataka kao što je to slučaj kada je u pitanju program BDE, nije potrebno da promenite komponente koje prepoznaju podatke, ali povežite komponentu DataSource sa `IBQuery1`. Kako ne koristim BDE,

potrebno je da unesem putanju InterBase baze podataka. Ipak, nemaju svi na svetu direktorijum Program Files, što zavisi od lokalne verzije Windowsa, a, naravno, Borland fajlovi sa primerima podataka se mogu instalirati bilo gde na disku. Ove probleme ćemo rešiti u narednom primeru.

UPOZORENJE

Primetićete da sam ja u kod ugradio i lozinku, veoma naivan način zaštite. Ne samo da svako može pokrenuti program, već neko može i izdvojiti lozinku ukoliko pogleda heksadecimalni kod izvršnog fajla. Ja sam koristio ovaj pristup da ne bih morao da stalno unosim lozinku dok testiram program, ali u pravoj aplikaciji trebalo bi da od korisnika zahtevate da unesu lozinku ukoliko im je stalo do sigurnosti podatka. ■

Izrada aktivnog upita

Primer IbEmpl2 je sadržao upit koji nije dozovljavao editovanje. Da biste aktivirali editovanje, potrebno je da upotrebite komponentu IBTable ili dodate upit u komponentu IBUpdateSQL, čak i kada je upit sasvim jednostavan. BDE obično uradi sav posao u pozadini koji Vam omogućava da izmenite rezultujući skup podataka jednostavnog upita, ali mi sada ne koristimo BDE.

Relacija između komponenata IBQuery i IBUpdateSQL je ista kao i između komponenata Query i UpdateSQL. Da bih to istakao, uzeo sam glavni formular primera UpdateSql koji sam pokazo ranije u ovom poglavlju i preneo sam na njega InterBase Express komponente izrađujući primer UpdSql2. Samo sam kopirao dve komponente iz originalnog primera, smestio ih na editor, promenio tip objekta i kopirao rezultujući tekst na novi formular. Svojstva su toliko slična, da sam morao da ignorišem nekoliko svojstava koja nedostaju (svojstva `DatabaseName` i `UpdateMode`).

Sada sam jednostavno dodao komponente IBDatabase i IBTransaction, izvor podataka i tabelu, i moj program je bio spreman i izvršavao se. Ključni element ovih komponenata je zapravo njihov SQL kod, koji je pridružen SQL svojstvu upita, i svojstva `ModifySQL`, `DeleteSQL` i `InsertSQL` komponente za ažuriranje.

Ipak, ovoga puta sam načinio referencu na bazu podataka nešto fleksibilnijom. Umesto da sam uneo naziv baze podataka u vreme dizajniranja, ja sam je izdvojio iz Windows Registryja (u koji je Borland beleži prilikom instaliranja programa). Ovo je kod koji se izvršava kada se program pokrene:

```
uses
  Registry;

procedure TForm1.FormCreate(Sender TObject);
var
  Reg: TRegistry;
begin
  Req := TRegistry.Create
  try
    Reg.RootKey := HKEY_LOCAL_MACHINE;
    Req.OpenKey(
    '\Software\Borland\Borland Shared\Data', False);
    IBDatabase1.DatabaseName :=
      Reg.ReadString('Rootdir') + '\employee.gdb';
  finally
    Req.CloseKey;
```

```
    Reg.Free;
  end;
  EmpDS.DataSet.Open;
end;
```

Ovo je, zapravo, veoma lep primer upotrebe `TRegistry` klase VCL-a, teme koju ću ponovo kratko obraditi u Poglavlju 19.

Nova karakteristika ovog primera, u poređenju sa prošlom verzijom, jeste postojanje komponente za transakciju. Kao što sam već rekao, InterBase Express komponente čine obaveznom upotrebu komponente za transakciju. Jednostavno, dodavanje nekoliko kontrola formularu za upisivanje i poništavanje transakcije bilo bi dovoljno, jer transakcija automatski započinje kada menjate bilo koji skup podataka koji joj je pridružen.

Takođe, poboljšao sam program dodavanjem komponente ActionList. Ovim se uključuju sve standardne akcije baze podataka i dodaju se dve akcije za podršku transakcije, Commit i Rollback. Obe akcije se aktiviraju kada je transakcija aktivna:

```
procedure TForm1.ActionUpdateTransactions(Sender: TObject);
begin
  acCommit.Enabled := IBTransaction1.InTransaction;
  acRollback.Enabled := acCommit.Enabled;
end;
```

Kada su izvršene, obavljaju glavnu operaciju, ali je potrebno da se ponovo otvori skup podataka u novoj transakciji (što se, takođe, može učiniti "zadržavanjem" konteksta transakcije):

```
procedure TForm1.acCommitExecute(Sender: TObject);
begin
  IBTransaction1.CommitRetaining;
end;

procedure TForm1.acRollbackExecute(Sender: TObject);
begin
  IBTransactionl.Rollback;
  // reopen the dataset in a new transaction
  IBTransaction1.StartTransaction;
  EmpDS.DataSet.Open;
end;
```

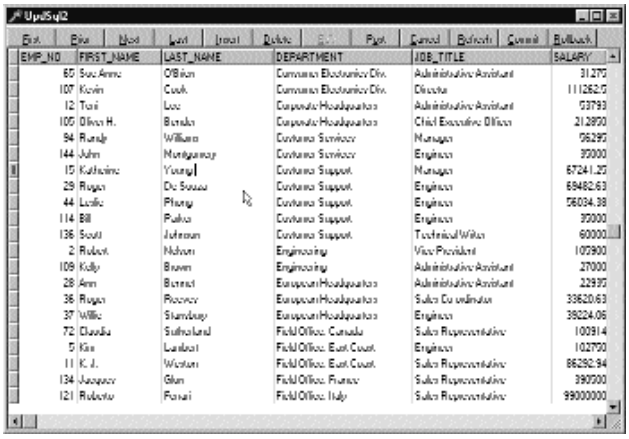
UPOZORENJE

Imajte na umu da InterBase zatvara bilo koji otvoreni kursor kada se transakcija završi, što znači da ponovo morate da otvorite kursore i ponovo pozovete podatke, čak i kada niste načinili nikakvu izmenu. Kada upisujete podatke, možete zatražiti od InterBasea da zadrži "kontekst transakcije" — da ne zatvori otvorene skupove podataka — zahtevanjem komande `CommitReatining`. U narednoj verziji 6.0 InterBasea moći ćete da koristtite komandu `RollbackRetaining`. Razlog za ovakvo ponašanje zavisi od činjenice da odgovarajuća transakcija odgovara snimku podataka. Kada se transakcija završi, pretpostavlja se da ponovo morate da pročitate podatke da biste dobili slogove koje su drugi korisnici možda promenili. ■

Poslednja operacija se odnosi na generički skup podataka, a ne na neki određeni, jer ja nameravam da dodam drugo rešenje za skup podataka programa. Akcije su povezane sa tekstualnom paletom alata, kao što možete videti na slici 11.10. Program otvara skup podataka pri-

likom pokretanja i automatski zatvara aktuelnu transakciju na izlasku iz programa, posle pitanja šta da učini, sledećom obradom događaja `Onclose`:

```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
  var
    nCode: Integer;
begin
  if IBTransaction1.rnTransaction then
  begin
    nCode := MessageDlg ('Commit Transaction? (No to rollback)',
      mtConfirmation, mbYesNoCancel, 0);
    case nCode of
      mrYes: IBTransaction1.Commit;
      mrNo: IBTransaction1.Rollback;
      mrCance1: Action := caNone; // don't close
    end;
  end;
end;
```



SLIKA 11.10 *Izlaz primera UpdSql2*

Umesto da koristite komponente IBQuery i IBUpdateSQL, možete koristiti komponentu IBDataSet, koja kombinuje ove dve komponente. InterBase skup podataka predstavlja aktivan upit sa potpunim skupom SQL iskaza za sve glavne operacije. Razlika između upotrebe dveju komponenata i jedne komponente je minimalna. Upotreba komponenata IBQuery i IBUpdateSQL je verovatno bolja kada prebacujete postojeću aplikaciju baziranu na dve ekvivalentne BDE komponente, iako prebacivanje programa direktno na komponentu IBDataSet ne zahteva više posla.

U primeru UpdSql2 ja sam upotrebio oba rešenja, tako da sami možete proveriti razlike. Evo dela DFM opisa komponente za skup podataka:

```
object IBDataSet1: TIBDataSet
  Database = IBDatabasel
  Transaction = IBTransaction1
  CachedUpdates = False
  BufferChunks = 32
  DeleteSQL.Strings = (
    'delete from EMPLOYEE'
    'where'
    ' EMP_NO = :OLD_EMP NO')
  InsertSQL.Strings = (
    'insert into EMPLOYEE'
    ' (FIRSTNAME , LASTNAME , SALARY, DEP_NO, JOB_ CODE,'
    ' JOB_GRADE, JOB_COUNTRY)'
    'values'
    ' (:FIRSTNAME, :LASTNAME, :SALARY, :DEPT_NO, :JOB CODE,'+
    ' :JOB_GRADE, :JOB_COUNTRY)')
  SelectSQL.Strings = (...)
  UpdateRecordTypes = [cusUnmodified, cusModified, cusInserted]
  ModifySQL.Strings = (...)
end
```

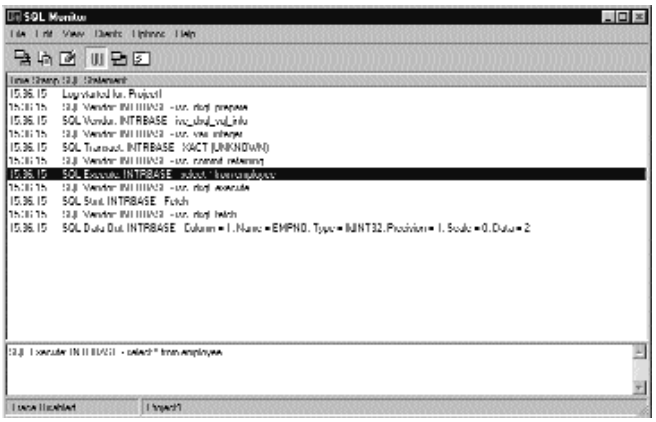
Ukoliko povežete komponente `IBQuery1` ili `IBDataSet1` sa izvorom podataka i pokrenete program, videćete da je njihovo ponašanje identično. Ne samo da komponente imaju sličan efekat, već su i svojstva i događaji koje možete koristiti veoma slični.

Klijent/server optimizacija

Kao što Vam je potrebno debagovnje da biste testirali Delphi aplikaciju (tema koju ću obraditi u Poglavlju 18), potrebni su Vam neki alati da biste testirali kako se klijent/server aplikacija ponaša i kako da je ubrzate ukoliko je moguće. Veoma je važno pogledati informacije koje se prebacuju od klijenta ka serveru (eksplicitne SQL zahteve koje radi naš program i one koje je dodao BDE) i od servera ka klijentu (podaci). Zbog toga je u Delphi Enterprise uključen alat SQL Monitor.

Upotreba SQL monitora

Kao što možete videti sa slike 11.11, centralni prozor SQL Monitora prikazuje listu komandi niskog nivoa koje su poslate serveru. Donji deo prozora prikazuje selekotovanu liniju iz liste iznad u više redova, što pomaže kada je linija previše dugačka.



SLIKA 11.11 *SQL Monitor prilikom izvršavanja*

Da biste upotrebili SQL Monitor, selektujte klijent program koji želite da proverite. Zatim pravilno podesite opcije (upotrebom odgovarajućih kontrola ili komandom Options➥Trace Options). Opcije koje su Vam na raspolaganju su navedene u tabeli 11.2.

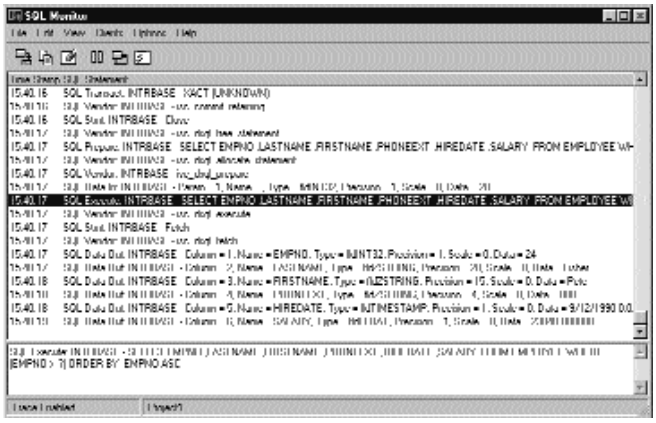
Tabela 11.2: SQL Monitorove opcije za praćenje.

Opcije za praćenje	Značenje
Prepared Query Statement	Omogućava praćenje SQL iskaza svaki put kada se pripremaju.
Executed Query Statement	Prati sve SQL iskaze koji se šalju serveru.
Input Parameters	Prikazuje ulazne parametre čim postanu dostupni. Ovo je važno za testiranje korektnosti parametara.
Fetched Data	Prikazuje podatke koje je poslao server (veoma spora operacija).
Statement Operations	Prikazuje zahteve koji su prethodili izvršavanju SQL iskaza, kao što su alokacija, priprema i izlaz.
Connect/Disconnect	Prikazuje događaje povezivanja i isključivanja. Ovo je važan test kada je određena vrednost `False` za `KeepConnection` `Database` komponente, jer klijent neće održavati vezu sa serverom, već će je uspostavljati samo onda kada je potrebna (sporedni efekat je smanjivanje potrebnih licenci). Ukoliko prikažete učestalost ovih zahteva, možete lakše odlučiti da li je bolje održavati vezu ili ne.
Transactions	Prati transakcije, uključujući i transakcije koje BDE automatski aktivira ukoliko transakcije ne koristite direktno.
Blob I/O	Prikazuje podatke o BLOB poljima.
Miscellaneous	Prati ostale operacije koje ne pripadaju ni jednoj od do sada navedenih kategorija.
Vendor Errors	Prikazuje poruke o grešci servera.
Vendor Calls	Prikazuje klijentove API pozive.

SQL monitor je koristan za proveru toga da li su SQL iskazi poslati od strane BDE-a serveru korektni, ali Vam, takođe, pomaže da vidite kako se obavljaju mnoge operacije. Uz vremenske informacije (time-stamp information) za svaku operaciju, brojne operacije mogu dati procenu brzine Vaše aplikacije (mada nemojte zaboraviti da izvršavanje SQL Monitora prilično usporava vezu).

Drugim rečima, SQL Monitor bi trebalo da bude Vaš vodič pri odluci kako da ubrzate Vašu klijent/server aplikaciju, upotrebom nekih od trikova koji su opisani u ovom poglavlju. Istovremeno, potrebno je dosta iskustva i dobro poznavanje SQL-a da biste pravilno interpretirali izlaz.

Kao primer upotrebe SQL Monitora možemo testirati šta se dešava kada koristimo svojstvo `Filter` komponente Table. U novom projektu upotrebite komponente Table, DataSource i DBGrid. Odaberite bazu podataka i tabelu (na primer, tabelu Employee lokalne baze podataka IBLocal) i odredite vrednost `True` za svojstvo `Filtered`, a za svojstvo `Filter` unesite `EmpNo > 20`. Ukoliko pokrenete program, SQL Monitor će prikazati da iskaz `select`, koji je generisao BDE, sadrži klauzulu `where` koja odgovara filtru. Ovo možete videti na slici 11.12.



SLIKA 11.12 *SQL Monitor prikazuje SQL iskaze koje je generisala komponenta Table*

Nadgledanje InterBase Expressa

SQL Monitor funkcioniše korišćenjem veze unutar BDE arhitekture. Zbog toga ga ne možete koristiti uz aplikacije koje se zasnivaju na InterBase Express komponentama. Umesto toga, u svoje aplikacije možete da umetnete kopiju komponente IBSQLMonitor i da načinite dnevnik.

Možete čak napisati generičku aplikaciju za nadgledanje, kao što sam ja to učinio u primeru IbxMon. Ja sam na formular primera postavio nadgledanje i kontrolu RichEdit, i napisao sam sledeću obradu `OnSQL` događaja:

```
procedure TForm1.IBSQLMonitor1SQL (EventText: String);
begin
  if Assigned (RichEdit1) then
    RichEdit1.Lines.Add (TimeToStr (Now) + ': ' + EventText);
end;
```

Test `if Assigned` može biti koristan kada se prima poruka tokom obaranja sistema, a dobija se kada napišete ovaj kod direktno unutar aplikacije koju nadgledate.

Da biste dobili poruku od drugih aplikacija (ili iz one koja se trenutno izvršava), potrebno je da uključite opcije praćenja komponente IBDatabase. U primeru UpdSql2 (koji je ranije objašnjen u sekciji "Izrada aktivnog upita") ja sam ih sve uključio:

```
object IBDatabase1: TIBDatabase
  ...
  TraceFlags = [tfQPrepare, tfQExecute, tfQFetch, tfError, tfStmt,
    tfConnect, tfTransact, tfBlob, tfService, tfMisc]
```

Ukoliko istovremeno pokrenete oba primera, izlaz u program IbxMon će prikazati spisak detalja o interakciji programa UpdSql2 sa InterBaseom, kao što možete videti na slici 11.13.

SLIKA 11.13 *Izlaz primera IbxMon, zasnovan na komponenti IBMonitor*

Podešavanje performansi

Pored upotrebe SQL Monitora za određivanje potencijalnih uskih grla Vaše aplikacije, postoji nekoliko stvari koje možete učiniti da biste ubrzali svoje klijent/server programe. Ključni element koji treba imati na umu — o kojem sam brinuo u ovom poglavlju— jeste smanjenje mrežnog saobraćaja, što se postiže smanjenjem veličine rezultujućih skupova podataka koje šalje server, kako u broju slogova tako i u veličini svakog sloga.

Pored sveukupnog dobrog dizajniranja baze podataka i njene dobre Delphi implementacije, postoje mnoge stvari koje treba proveriti. Sledeći saveti Vam mogu pomoći, ali Vam neće pomoći toliko kao dobar dizajn baze podataka.

- U InterBaseu možete podesiti interval automatskog čišćenja (ili "sakupljanje otpada"). Operacija se takođe automatski obavlja prilikom pravljenja rezervne kopije. Kako čišćenje usporava bazu podataka, ne treba ga previše često obavljati.

Ipak, ukoliko ga nikada ne obavite, baza podataka će voditi evidenciju o mnogim zaostalim uklonjenim slogovima, čime će se smanjiti performanse i koristiće se više memorije.

- Koristite indekse što je češće moguće, naročito ukoliko po njima sortirate rezultujući skup podataka. Imajte na umu da će dobar RDBMS umesto Vas dodati bar privremene indekse. Upotreba indeksa može prilično da ubrza upite, naročito ukoliko se indeksirana polja koriste za spajanje tabela.
- Ukoliko polja soritrate u opadajućem redosledu, odgovarajući opadajući indeks može biti od pomoći.
- Ukoliko ste iskusan korisnik, možete proučiti plan upita (query plan), pristup koji koristi server za izvršavanje upita, koji se prikazuje (na primer) kada koristite WISQL. Plan upita će Vam pokazati da li SQL server koristi indekse. U nekim slučajevima ćete, možda, morati da izmenite neke složene upite da biste pomogli optimizatoru ugrađenom u RDBMS.
- Proverite podešavanja servera, izbegavajte prekomernu upotrebu transakcija i pokušajte da budu kratke i usredsređene. Koristite keširana ažuriranja umesto transakcija (ili zajedno sa njima) da biste omogućili klijent kompjuteru da za Vas obavi deo posla, i izbegavajte neke skupe server operacije.
- Direktno obrađujte transakcije, isključujući karakteristiku auto-commit BDE-a; da biste to učinili, odredite `SQLPASSTHRU MODE` za `SHARED NOAUTOCOMMIT`. (Ove i druge karakteristike BDE-a su opisane u listi uz program BDE Administrator.)
- Ukoliko nemate probleme sa licencama, odredite vrednost `True` za svojstvo `KeepConnection` komponente Database.
- Odredite vrednost `0` za `TRACE MODE` kada ne debagujete, da biste izbegli da drajver šalje stringove praćenja debageru i time usporava operacije. Kada radite sa InterBase Express komponentama, pozovite `DisableMonitoring` iz koda za inicijalizaciju da biste isključili praćenje.
- Uključite keširanje šeme (odredite `TRUE` za `ENABLE SCHEMA CACHE`). Ova vrednost smanjuje vreme potrebno za otvaranje tabele jer klijent ne mora da traži metapodatke. Takođe, možete upotrebiti svojstva Delphi `FieldDefs` i `StoredDefs` komponente Table za čuvanje metapodataka direktno u klijent programu.
- Kada koristite Microsoft i Sybase SQL servere, pokušajte da parametar `PACKETSIZE` smanjite na minimum od 4K, takođe menjajući odgovarajuću vrednost na serveru. Kod ovih servera takođe proverite da li je za parametar `DRIVER FLAGS` određena vrednost `0`. Ukoliko je određena vrednost `2048`, upiti će se izvršavati u asinhronom modu i biće mnogo sporiji.
- Kada koristite Oracle, DB/2 i ODBC drajvere, pokušajte da fino podesite parametar `ROWSET SIZE` tako da dobijete najbolje performanse.

- Kada koristite InterBase drajver, ukoliko eksplicitno ne koristite transkacije, odredite vrednost `4096` za parametar `DRIVER FLAGS`. Ova vrednost omogućava "meka upisivanja" (soft commits), što znači da se posle svake operacije upisivanja ili poništavanja otvoreni kursor ne mora osvežavati.

Šta je sledeće?

Ovo poglavlje je predstavilo uvod u klijent/server programiranje upotrebom Delphija. Upoznali smo se sa ključnim elementima, upoznali smo se sa najvažnijim karakteristikama SQL jezika i malo smo se bavili nekim interesantnim oblastima klijent/server programiranja. Potpuno razmatranje klijent/server programiranja u Delphiju bi verovatno zahtevalo jednu zasebnu knjigu.

Isto se može reći za ADO programiranje, koje se samo ukratko predstavlja u narednom poglavlju. ADO je interfejs za mehanizam baze podataka koji je Microsoft promovisao (nazvan OLE DB). Pošto radimo na Windows platformi, potrebno je da znamo bar nešto o ADO-u, bez obzira na to koju bazu podataka koristimo.

POGLAVLJE

12

Upotreba ADO-a

KADA JE DELPHI PRVOBITNO PREDSTAVLJEN, BORLAND JE POSEDOVAO PARADOX I VISUAL DBASE I ZBOG TOGA JE IZRADIO JEDAN MEHANIZAM ZA PRISTUP PODACIMA KOJI SU KORISTILI SVI NJEGOVI PROIZVODI. TAJ MEHANIZAM JE BDE, A JA SAM GA RAZMATRAO U POSLEDNJA TRI POGLAVLJA. BDE JE DIZAJNIRAN ZA POVEZIVANJE SA MNOGIM SQL SERVERIMA, UKLJUČUJUĆI ORACLE I INTERBASE, I ZA PRISTUPANJE DRUGIM IZVORIMA PODATAKA PREKO ODBC-A, PRVOG MICROSOFTOVOG INTERFEJSA ZA PRISTUPANJE PODACIMA. ODBC JE PRVOBITNO BIO VEOMA SPOR, A NEKI OD DRAJVERA ZA SPECIFIČNE FORMATE BAZE PODATAKA NISU NAPRAVLJENI BEZ GREŠAKA.

Tokom vremena desile su se tri stvari. Prvo, Borland je prodao svoje korisničke proizvode za baze podataka. Drugo, Microsoftova uloga kao provajdera baza podataka je porasla kako su MS Access i MS SQL Server bivali sve više prihvaćeni. Bilo da se ovo desilo zbog tehničkih razloga, bilo da je Microsoft imao bolju reklamu, to sada nije važno. Kao Delphi programer Vi ne možete uvek birati tehnologiju kojom ćete raditi, a šanse da će Vaši programi raditi sa Microsoftovim bazama podataka su velike. Treće, Microsoft je poboljšao strategiju pristupa podacima uvodeći DAO, RDO, OLE DB i ADO. Šta znače ove skraćenice, videćemo u narednom odeljku.

Borland je prihvatio ove promene tokom nekoliko poslednjih godina dodavanjem specifičnih BDE drajvera za MS Access. Koristeći BDE moguće je povezati se sa Access bazom podataka, ali to implicira upotrebu dva mehanizma baza podataka odjednom, dok pristup Accessu preko ADO-a predstavlja direktniju vezu. Nije bilo mnogo toga što je Borland mogao da učini jer Microsoft nije bio dovoljno otvoren u dokumentovanju upotrebe svojih mehanizama baza podataka, naročito kada je u pitanju JET Engine koji koriste Visual Basic programeri za povezivanje sa Access bazama podataka. Dalje, JET Engine ne manipuliše podacima konzistentno kao što to čini BDE, čime je otežana istovremena upotreba ovih dvaju mehanizama.

Microsoft ActiveX Data Objects (ADO), novi interfejs za podatke visokog nivoa, ponovo je promenio situaciju, a Delphi 5 sada podržava ovu tehnologiju direktno koristeći specifične DataSet komponente. Ove komponente ne koriste BDE i ne zahtevaju posebno instaliranje na kompjuterima klijenata. Umesto toga, ove komponente zahtevaju postojanje Microsoft ADO i OLE DB mehanizama baze podataka na klijent mašinama. Kako se ADO i OLE DB po definiciji instaliraju sa Windowsom 2000 i Windowsom 98, glavobolje oko instaliranja mehanizma baze podataka uz Vašu aplikaciju će se u budućnosti verovatno smanjiti, jer će ove platforme postepeno zameniti starije operativne sisteme. Ipak, Windows 95 i Windows NT 4 nemaju obavezno instaliran ADO, te ćete verovatno morati da distribuirate i instalirate ADO mehanizam uz Vašu aplikaciju, ili ćete morati da zahtevate od korisnika da instalira ADO pre instaliranja Vaše aplikacije, da bi ona mogla da se izvršava na starijim platformama. Čak i kada je ADO već instaliran na sistemu, Vaš proces instaliranja će verovatno morati ponovo da konfiguriše ADO za potrebe Vaše aplikacije. Upotreba ADO komponenata ne zasenjuje toliko BDE instalaciju i konfigurisanje koliko to čini zamena instalijom ADO-a. Za dalje informacije posetite www.microsoft.com/data.

Pre nego što razmotrimo Delphi 5 ADO komponente, dozvolite mi da rekapituliram ključne koncepte sakrivene iza Microsoftovih tehnologija podataka.

Microsoftov put ka podacima

Molim Vas da ostanete uz mene čak i ukoliko Vam ovaj odeljak bude više ličio na Microsoftovu reklamu, u kojoj je više skraćenica nego tehničkih detalja, ali kada izrađujete programe za Windows, morate živeti u Microsoftovom svetu.

Microsoftova strategija za obezbeđivanje pristupa bilo kojoj vrsti podataka naziva se *Universal Data Access* (univerzalni pristup podacima). Ideja je da postoji jedan interfejs koji omogućava programerima upotrebu njihovih omiljenih alata za pristup relacionim bazama podataka i drugim manje struktuiranim izvorima podataka (kao što su elektronske poruke i radne tabele).

Universal Data Access je ideja, a u praksi se ova strategija ostvaruje instaliranjem Microsoft Data Access Components (MDAC — Microsoftove komponente za pristup podacima) na Win32 sistemu. Windows 2000 će MDAC imati kao deo operativnog sistema, ali se ove komponente takođe mogu preuzeti sa Microsoftovog web sajta, `www.microsoft.com`.

NAPOMENA

Delphi 5 CD sadrži instalaciju MDAC-a koji je potrebno instalirati da biste mogli da koristite ADO komponente, izuzev ukoliko ADO podršku već nemate na Vašem sistemu. ■

Sada, da bi stvari bile još zamršenije, MDAC sadrži ActiveX Data *Objects* (ADO), OLE DB i Open Databse Connectivity (ODBC) podršku.

ADO i OLE DB

ADO obezbeđuje API koji programeri moraju da pozivaju da bi izradili rešenja prema Microsoftovoj strategiji. ADO je dizajniran sa ciljem da bude jedini interfejs podataka koji je potreban za bilo koji programerski zadatak. Ja zaista sumnjam da će svi Delphi programeri preći na ADO, ali je to svakako tehnologija koju ne treba potcenjivati.

Kada bih ja bio osoba koja predstavlja Microsoft, rekao bih "da su glavne prednosti ADO-a lakoća upotrebe, velika brzina, malo opterećenje memorije, male potrebe za prostorom na disku, minimalni mrežni saobraćaj u ključnim situacijama i minimalni broj slojeva između aplikacije i skladištenja podataka — sve da bi se obezbedio interfejs visokih performansi". Ovo je citirano iz Microsoftove literature o ADO-u. ADO predstavlja alternativu za BDE koristeći COM interfejs, mada, kada upotrebljavate Delphi 5 ADO komponente, neće Vam biti potrebno da razumete COM.

Dok ADO obezbeđuje interfejs, pristup podacima obavlja drugi sloj nazvan OLE DB. Ovo je programski interfejs na nivou sistema i može se smatrati naslednikom ODBC-a, koji je još uvek na raspolaganju kao alternativni način pritupanja podacima. OLE DB proširuje ODBC obezbeđujući pristup izvorima podataka koji nisu relacioni, uključujući baze podataka glavnih računara i hijerarhijske baze podataka, kao i elektronskoj pošti i sistemu čuvanja fajlova.

Možda se pitate zašto je Microsoft predstavio ADO sloj umesto da programerima dopusti pristup OLE DB-u. Osnovni razlog je složenost OLE DB-a. ADO enkapsulira preko šezdeset OLE DB interfejsa, koje nije lako direktno povezati, u oko 20 jednostavnih objekata. Kada koristite ADO, ne morate brinuti o interfejsima, alokaciji memorije, prebrojavanju referenci ili klasama (a to se sve tiče OLE DB-a).

Drugi odgovor je da je ADO namenjen programerskim alatima visokog nivoa (kao što je Visual Basic) koji ne mogu da se nose sa COM Interfaceima preko nižeg nivoa VTables. Umesto toga, Microsoft je kreirao neke ActiveX objekte koji sadrže OLE DB funkcionalnost, prikazujući programerima samo najvažnije koncepte na jednostavniji i lakši način. Upotrebom ADO-a, svaki alat koji podržava ActiveX tehnologiju (bilo da su to kompajleri ili jednostavni jezici za skriptove), može zaista da spoji UDA model. Zapravo, Vi možete napisati ADO programe bez specifičnih komponenata koje obezbeđuje Delphi 5. Ipak, ADO komponente koje možete upotrebiti, omogućavaju Vam da programirate ADO koristeći iste tehnike koje su Vam već poznate.

NAPOMENA

Evo još terminologije: OLE DB se može posmatrati kao skup komponenata koje predstavljaju provajdere podataka, koje sadrže i prikazuju podatke, kao korisnike podataka i kao servisne komponente, koje obrađuju i prosleđuju podatke (kao što su procesori upita i mehanizmi kursora). ■

ADO objekti

ADO arhitektura je izgrađena oko nekoliko obejkata za koje Delphi obezbeđuje komponente koje ih sadrže. Evo kratke liste ključnih ADO objekata:

- Objekat *Connection* nudi načine za pristup izvoru podataka, upotrebom stringova za povezivanje (*connection strings*) za pronalaženje provajdera podataka, upravljanje odgovaajućom sesijom i upravljanje transakcijama.
- Objekat *Command* Vam omogućava da radite na izvoru podataka (kojim upravlja objekat Connection), dajući Vam mogućnost upotrebe upita, omogućavajući Vam dodavanje, uklanjanje i ažuriranje podataka. Objekat Command može da sadrži jedan ili više *Parameter* objekata. Među bazama podataka koje nisu relacione, a koje ADO podržava, različiti provajderi podataka podržavaju različite komandne stringove (i ne podržavaju svi SQL komande).
- Objekat *Recordset* je rezultat komande Query, to jest, keš slogova koje daje upit. Recordset omogućava kretanje i izmenu podataka. Svaki red Recordseta se sastoji iz više *Field* objekata.

Ostali ADO objekti sadrže Error objekat i Errors kolekciju, Property objekte (koji sadrže svojstva objekata i dinamička svojstva koja su im pridružena u vreme izvršavanja) i Properties kolekciju. Konačno, postoje tri događaja koja predstavljaju notifikaciju da operacija treba da se odigra ili je obavljena.

Čitajući ovaj kratak opis, verovatno ste shvatili da se ADO arhitektura ne razlikuje mnogo od Delphi arhitekture skupa podataka. Čak i pored sličnosti zaista postoje i mnoge razlike između ADO objekata i Delphi komponenata. Zbog toga, mada je moguće ove objekte direktno programirati, Delphi 5 obavija ADO objekte poznatim komponentama dajući jednostavno rešenje. Mapiranje ADO koncepata u VCL koncepte verovatno nije tako jednostavno, uzimajući u obzir da ADO DB jedinica VCL-a, koja je za ovo odgovorna, sadrži više od 5000 linija izvornog koda.

NAPOMENA

Delphi 5 ADO komponente su sadržane u verziji Delphi Enterprise i vlasnici Delphi Professionala ih mogu dokupiti. ■

Delphi 5 ADO komponente

Delphi ADO komponente su izvedene iz klase `TDataSet`. Ključni dobitak je integracija sa Delphi IDE-om i mogućnost upotrebe svih kontrola koje su unapred definisane i kontrola koje obezbeđuju nezavisne programerske kuće, a koje služe za prikazivanje i editovanje podataka.

Postoji nekoliko ADO DataSet komponenata u Delphiju 5. Postoje tri komponente koje možete koristiti kada pišete aplikacije za baze podataka:

- Komponenta `TADOConnection` obavija objekat ADO Connection, obezbeđujući string za povezivanje, prijavljivanje i transakcije. Karakteristike ove komponente su slične karakteristikama komponente `TDatabase`.
- Komponenta `TADOCommand` obavija objekat ADO Command, obezbeđujući način za izvršavanje upita koji kao rezultat ne daje skup podataka.
- Komponenta `TADODataSet` obavija istovremeno objekte ADO Command i ADO Recordset. Kao što je slučaj i sa svakim skupom podataka, Vi navodite komandu koja se odnosi na jednu tabelu, ili pretražujete više tabela i kao rezultat dobijate skup slogova.

Ove komponente su sve što Vam je potrebno da biste koristili ADO u Delphi aplikacijama. Ipak, način na koji ih koristite se prilično razlikuje od tradicionalnog Delphi programiranja kad se koriste komponente `TTable` i `TQuery`. Svojstva koja su Vam na rspolaganju se prilično razlikuju, pa tako i programerski stil. Kako bi ovo otežalo prebacivanje postojećih programa na ADO, Delphi 5 takođe sadrži neke specijalizovane ADO komponente za pristupanje podacima, čija svojstva i karakteristike odgovaraju standardnim DataSet komponentama. Te tri komponente su `TADOTable`, `TADOQuery` i `TADOStoredProc`.

NAPOMENA

Pored novih komponenata za skupove podataka, ADO podrška u Delphiju 5 sadrži i neke nove tipove polja, što sam pomenuo u Poglavlju 9. Specifične klase polja uključuju `TWideStringField` za stringove koji su bazirani na Unicode karakterima; `TGuidField` za čuvanje jedinstvenih globalnih identifikatora (Globally Unique Identifiers); `TVariantField` za polja koja sadrže Variant tip podataka (to jest, za polja za koja nije određen tip podataka); `TInterfaceField` i izvedeni `TIDispatchField`; generički COM interfejs i COM `IDispatch` interfejs. ■

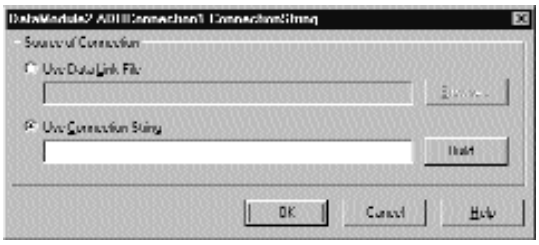
Praktični ADO primer

Sada kada ste se upoznali sa ključnim konceptima i komponentama koje možete koristiti za ADO programiranje, vreme je da pogledamo kod. Ja ću prvo izraditi veoma jednostavan primer, nazvan AdoPrimer, koji koristi ODBC vezu za pristupanje tabeli dBase baze podataka DBDEMOS. Program se zasniva na modulu podataka kojem sam dodao komponentu ADOConnection. Zapravo, komponenta za povezivanje nije neophodna jer možete koristiti svojstvo `ConnectionString` komponente ADODataSet.

UPOZORENJE

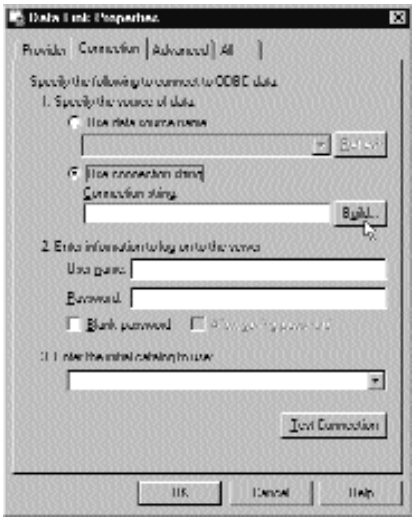
Kada koristite `ConnectionString` umesto povezivanja skupa podataka sa ADOConnection, znači da će Recordset kreirati novi string za povezivanje. U tom slučaju, ukoliko otvorite više skupova podataka iz istog programa, dobićete više veza, čime tračite resurse i, moguće, licence za pristup klijenta. Ukoliko nameravate da dobijete više od jednog Recordseta od ADO izvora, potrebno je da uvek koristite objekat ADOConnection koji je deljiv. ■

Podešavanje povezivanja zahteva da naznačite kako ćete pristupati podacima i gde se oni fizički nalaze. Svojstvo `ConnectionString` komponente ADOConnection (kao i komponente ADODataSet) aktivira za Vas specijalni editor koji možete videti na slici 12.1. Ovo je, takođe, unapred određeni editor za komponentu. U osnovi su Vam na raspolaganju dve mogućnosti: možete sami da unesete string za povezivanje, ili možete da upotrebite jedan iz Microsoft Data Link fajla (`.UDL`) i referišete se na fajl u stringu za povezivanje.



SLIKA 12.1 *Editor svojstva ConnectionString i komponente ADOConnection*

Ukoliko kliknete opcionu kontrolu Connection String i kliknete kontrolu Build, prikazaće se okvir za dijalog Data Link koji obezbeđuje Microsoft. Isti okvir za dijalog se aktivira kada editujete Data Link fajl. Okvir za dijalog Data Link sadrži više strana. Na prvoj strani (Provider) potrebno je da odaberete OLE DB provajdera. Prelaskom na narednu stranu (Connection), videćete različite elemente u zavisnosti od Vašeg izbora provajdera. Na slici 12.2 su prikazane moguće opcije za ODBC provajdera.



SLIKA 12.2 *Okvir za dijalog Data Link koji obezbeđuje Microsoft*

Ponoviću, možete odabrati tip izvora podataka (Use data source name) ili naznačiti bazu podataka (Use connection string). U drugom slučaju, ukoliko kliknete kontrolu Build, biće zatraženo da

odaberete tip izvora podataka i fajl ili direktorijum. Ja sam odabrao dBase i Delphi demos direktorijum. Konačno, možete izmeniti mrežne veze i dozvole pristupa na trećoj strani (Advanced) i proveriti da li je sve kako treba na poslednjoj strani (All). Vrednosti koje sam ja upotrebio u primeru daju sledeći string (koji je drugačije formatiran da bi bio čitljiv):

```
Provider=MSDASQL.1;
Persist Security Info= False;
Mode=Read¦Write;
Connect Timeout=15;
Extended Properties='"
  DSN=dBASE Files;
  DBQ=c:\PROGRAM FILES\COMMON FILES\BORLAND SHARED\DATA;
  DefaultDir=c:\PROGRAM FILES\COMMON FILES\
    BORLAND\
  SHARED\DATA;
  DriverId=533;
  MaxBufferSize=2048;
  PageTtimeout=5;";
Locale Identifier=1033
```

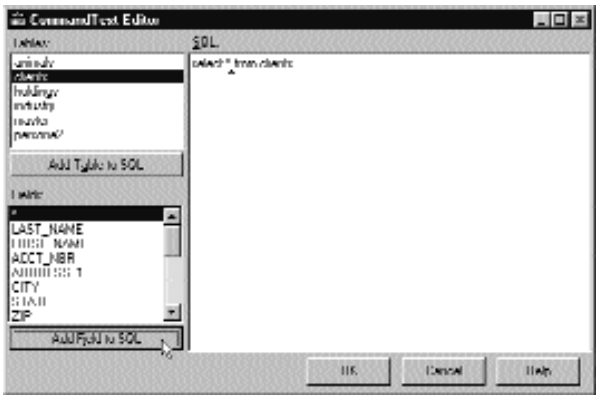
Kada je sve podešeno, možete testirati string za povezivanje ukoliko kliknete kontrolu na strani Connections okvira za dijalog Data Link, ili promenom vrednosti svojstva `Connected` odgovarajuće komponente. Ja sam, takođe, deaktivirao svojstvo `LoginPrompt` jer dBase nema podršku za lozinku.

Pošto smo podesili ADOConnection komponente, možemo dodati modulu podataka komponentu ADODataSet. Ova komponenta sadrži svojstvo `Connection` koje možete upotrebiti za referisanje na ADOConnection, što je alternativa podešavanju posebnog `ConnectionStringa`. Zatim, potrebno je da navedete komandu koju želite da pošaljete bazi podataka. Tip komande je naznačen svojstvom `CommnadType`, koje označava kako treba interpretirati svojstvo `CommandText`, koje sadrži komandu. Na primer, tabeli CLIENTS.DB možemo pristupiti upotrebom sledećih vrednosti:

```
object ADODataSet: TADODataSet
  Active = True
  Connection = ADOConnection
  CommandText = 'clients'
  CommandType = cmdTable
end
```

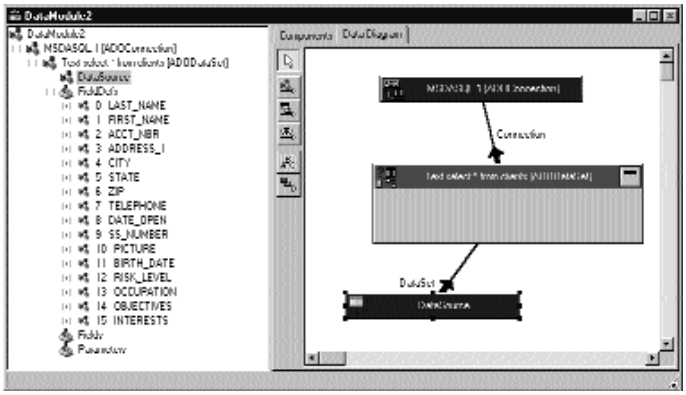
Takođe se možete direktno referisati na uskladištenu proceduru (tip komande `cmdStoredProc`), ali najčešće ćete koristiti tip komande `cmdText` i pisaćete SQL upit u komandnom tekstu. Ista dva svojstva CommandText i CommandType su Vam na raspolaganju za komponentu ADOConnection. Kao što sam ranije pomenuo, ADODataSet predstavlja specijalni tip ADO komande koja daje skup podataka.

Ukoliko planirate da koristite upit, možete upotrebiti editor svojstva CommandText, koji predstavlja prostu pomoć, što se može videti na slici 12.3. U ovom slučaju, ja sam odabrao tabelu i sva polja tabele. Primetićete da je ovaj editor prikazan čak i kada tip komande nije `cmdText`, što može dovesti do zabune.



SLIKA 12.3 *Editor svojstva CommandText komponenata ADODataSet i ADOCommand*

U ovom trenutku možemo jednostavno dodati komponentu izvora podataka modulu podataka, i proizvesti dijagram podataka koji je prikazan na slici 12.4. Sve ove komponente se nalaze u modulu podataka. Formular je sa njima povezan iskazom `uses` i sadrži komponente DBGrid i DBNavigator, koje su povezane sa izvorom podataka modula podataka.



SLIKA 12.4 *Dijagram podataka za demo program AdoPreimer*

Od Paradoxa do Accessa

Pre nego što se pozabavimo naprednijim karakteristikama, potrebno je da pogledamo primer koji smo izradili u prethodnim poglavljima i koji je prebačen u ADO. Prvi primer koji sam konvertovao je DbAware, iz Poglavlja 9, jer ovaj program kreira potpuno novu tabelu. Kasnije ću izraditi primer koji konvertuje postojeće tabele iz baze podataka DBDEMOS, i prebaciću i ostale programe koje sam izradio u prethodnim poglavljima.

NAPOMENA

Delphi 5 primer baze podataka zapravo sadrži bazu podataka Access MDB koja je slična bazi podataka DBDEMOS. Ipak, ja ću Vam pokazti postupak konvertovanja jer će biti potrebno da svoje programe prebacite konvertovanjem tabela. Ovo je takođe moguće uraditi upotrebom Paradox ODBC drajvera za pristup DBDEMOS tabelama koristeći ADO, ali smatram da je MS Access bolji izbor prilikom razmatranja ADO karakteristika. Nije potrebno da koristite ADO ukoliko planirate da koristite Paradox format fajlova baze podataka, jer BDE definiše standard za podršku Paradox formatu fajlova baze podataka. ■

Uopšte uzev, kada prebacujete postojeće aplikacije u ADO, možete imati koristi kada upotrebljavate komponente `TADOTable` i `TADOQuery`. Ove komponente nemaju potpuno ista svojstva kao odgovarajuće BDE komponente, ali ipak imaju neke zajedničke elemente.

ADO komponenta za tabele sadrži intuitivno svojstvo `TableName`, koje je mapirano na svojstvo `CommandText` ADO skupa podataka. Takođe obezbeđuje master-detail vezu između tabela (upotrebom svojstava `MasterSource` i `MasterField`).

ADO komponenta za upite sadrži intuitivno svojstvo SQL, koje je takođe mapirano na svojstvo `CommandText`, i `DataSource` za izradu master-detail strukture, ili neki drugi način za dobijanje vrednosti parametara iz nekog drugog skupa podataka.

Obe klase obezbeđuju nešto više od komponente `TADODataset`, koja, opet, obezbeđuje nešto više od komponente `TADOCustomDataset` iz koje su sve tri komponente izvedene. Ova poslednja klasa je mesto gde se obavlja sav posao za ADO skup podataka koji obezbeđuje Delphi.

Upotreba komponente ADOTable

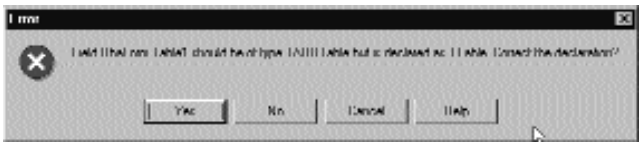
Posle ovog uvoda vratimo se na primer. Kao što sam rekao, ja sam konvertovao primer DbAware iz Poglavlja 9 u primer DbAware2. Jedina operacija koju sam morao da obavim bila je da otvorim originalni formular, otvorim DFM fajl i promenim liniju

```
object Table1: TTable
```

u

```
object Table1: TADOTable
```

Kada sačuvate ili kompajlirate konvertovani primer, Delphi će Vas pitati da li da izmeni tip objekta Table1 u Pascal fajlu, što je zgodna nova karakteristika koja se javlja kada promenite tip komponente. Delphijev zahtev je prikazan na slici 12.5.



SLIKA 12.5 *Kada promenite tip objekta u DFM fajlu, Delphi 5 Vas pita da li da konvertuje polje u Pascal kodu*

Konvertovanjem DFM fajla nazad u formular, videćete da Delphi ne prepoznaje svojstvo `Database`. To nije problem jer ADO tabela koristi svojstvo `Connection`. Ja sam programu dodao

komponentu za povezivanje koja pokazuje na fajl `MdData.mdb` koji se nalazi u Data direktorijumu primera ovog poglavlja. Usput, ja sam ovu vezu dobio upotrebom UDL fajla koji možete naći u Data direktorijumu.

UPOZORENJE

Svi fajlovi za povezivanje i stringovi za povezivanje primera programa ove knjige referišu se na svoje podatke upotrebom apsolutnih putanja. Ove putanje sadrže unapred određenu strukturu direktorijuma (fascikli) koja se kreira kada "raspakujete" arhivu koju ste preuzeli sa Sybexovog web sajta. Ukoliko programe premestite u neku drugu strukturu direktorijuma, potrebno je da prepravite ove veze da biste mogli da pokrenete primere. Svakako je moguće ispraviti ovaj problem određivanjem veze u vreme izvršavanja, ali u ovom slučaju nećete moći da radite sa aktivnim podacima u vreme dizajniranja. ■

Pravi problem je to što će se javiti još jedna greška koja se tiče definicije polja. ADO komponenta za tabele ne omogućava upotrebu definicija polja za kreiranje tabele, kao što je slučaj u originalnom primeru. Kao rešenje možemo da kreiramo tabelu (nad postojećom bazom podataka) koristeći SQL kod. Kao što smo videli u DDL (Data Definition Language) odeljku prethodnog poglavlja, možete da upotrebite SQL komandu za kreiranje tabele. Ja sam dodao komponentu ADOCommand, koja je "zakačena" na istu vezu, i uneo sam sledeći tekst u editor svojstva `CommandText`:

```
create table workers (
  firstname TEXT(30),
  lastname TEXT(30),
  department INTEGER,
  branch TEXT(20),
  senior YESNO,
  hiredate DATETIME);
```

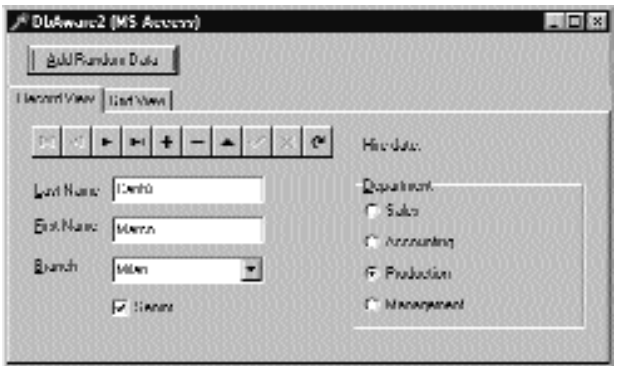
Tipovi podataka koji su ovde navedeni, takođe su na raspolaganju i u MS Accessu, ali to nisu SQL generički tipovi podataka. U narednoj tabeli ćete videti spisak MS Access tipova podataka:

Tip	Opis
TEXT (veličina)	String od najviše 255 karaktera.
MEMO ili LONGTEXT	String veličine najviše 1.2 gigabajta.
BYTE	Broj iz opsega od 0 do 255.
SHORT	Ceo broj (2 bajta).
LONG ili INTEGER	Ceo broj (4 bajta).
COUNTER	Celobrojna vrednost koja se automatski uvećava za naredni slog.
SINGLE	Broj u pokretnom zarezu veličine 4 bajta.
DOUBLE	Broj u pokretnom zarezu veličine 8 bajtova.
CURRENCY	Osmobitna numerička vrednost sa četiri decimalne cifre (kao i Delphi tip podataka za valute).
GUID	COM GUID (128 bitova)
DATETIME	Broj u pokretnom zarezu (DOUBLE) za datum i vreme.
BIT ili YESNO	Boolean vrednost ili jedan bit.
BINARY (veličina)	Binarni podaci naznačene veličine.
LONGBINARY	Veliki binarni podatak.

Posle podešavanja ove komande u stringu za povezivanje, promenio sam kod obrade `OnCreate` događaja formulara u sledeći kod:

```
procedure TDbaForm.FormCreate(Sender: TObject);
var
  TablesList:TStringList;
begin
  // read table names form database
  TablesList := TStringList.Create;
  try
    ADOConnection.GetTableNames (TableList);
    // check if the table already exists
    if TablesList.IndexOf (Table1.TableName) < 0 then
      // create it
      ADOCommand. Execute;
    // open the new or existing table
    Table1.Open;
  finally
    TablesList. Free;
  end;
end;
```

Da bi se proverilo da li tabela postoji, ova procedura koristi metod `GetTableNames` komponente za povezivanje, čime se dobijaju sve tabele i proverava da li tabela kojoj se obraćamo komponentom ADOTable postoji među tabelama. Ukoliko ne postoji, izvršava se komanda, što rezultuje kreiranjem tabele. Kada se ovo obavi, konvertovani primer se pravilno izvršava, kao što je to bio slučaj i sa originalnim primerom. Izlaz ovog primera je prikazan na slici 12.6, mada nema šta naročito da se vidi: sve razlike su zapravo iza kulisa.



SLIKA 12.6 *Primer DbAware2, koji koristi MS Access tabelu koja se kreira*

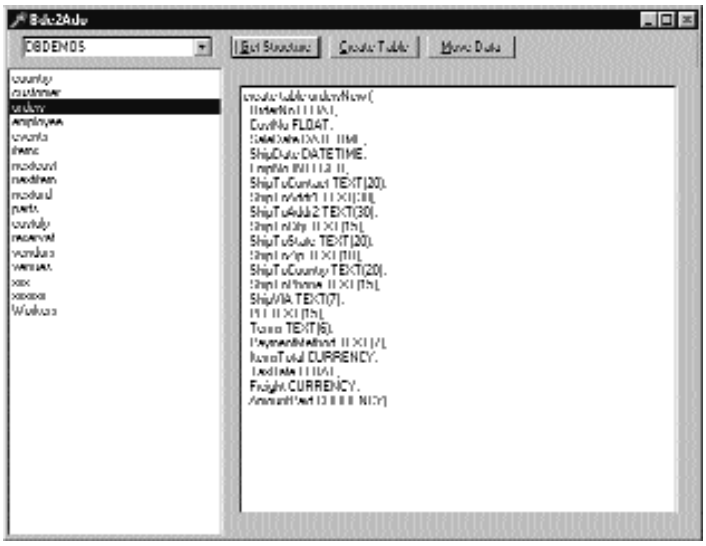
Kopiranje tabela

Da bih mogao da konvertujem neke druge BDE primere u ADO, ili da bar iskoristim njihov izvorni kod, odlučio sam da prebacim još tabela baze podataka DBDEMOS u Access bazu podataka. Pokazaću Vam proces konvertovanja umesto da koristim bazu podataka DBDEMOS koja je uključena u Delphi, jer će biti potrebno da Vaše programe prebacite konvertovanjem tabela. Ovaj primer daje smernice i kod koji možete upotrebiti. Jednostavan program koji sam napisao, Dbe2Ado, nije naročito interesantan iz perspektive korisnika, ali mi je omogućio da objasnim neke tehnike ili, bar, da generalizujem tehnike opisane u poslednjem odeljku.

Primer Bde2Ado je pozjamio inicijalni kod od primera Tables iz Poglavlja 9. Zapravo, prilikom pokretanja, popunjava se combo polje nazivima alijasa koji su na raspolaganju, a lista se popunjava tabelama odabrane baze podataka. Jedina razlika je što sada, u pozivu `GetTablesNames`, ja filtriram samo Paradox tabele (drugi parametar) i ne prikazujem ekstenzije (treći parametar):

```
Session.GetTablesNames (ComboBox1.Text, '*.db',
  False, False, ListBox1.Items);
```

Kao što možete videti na slici 12.7, operacije posle selektovanja tabele obavljaju se u tri ručna koraka. To su generisanje SQL iskaza Create Table (kontrola Get Structure), izvršavanje iskaza (kontrola Create Table) i kopiranje podataka (kontrola Move Data). Prva operacija čita definicije polja iz Paradox tabele i generiše SQL kod. Na ovaj način, ukoliko kod nije odgovarajući, Vi ga možete popraviti (kao što sam ranije pomenuo, ovo nije program za krajnje korisnike već za programere).



SLIKA 12.7 *Program Bde2Ado, kada je SQL iskaz za kreiranje generisan klikom na prvu kontrolu*

Obrada događaja za prvu kontrolu započinje određivanjem naziva tabele. Procedura koristi originalan naziv, izuzev ukoliko tabela već ne postoji, kada dodaje reč *new* nazivu tabele (jednom ili više

puta sve dok se ne dobije jedinstveni naziv tabele). Na primer, ukoliko već postoje tabela MyTable i tabela MyTableNew, program kreira tabelu MyTableNewNew:

```
procedure TForm1.btnGetStructureClick(Sender: TObject);
begin
  ...
  // find a new table name
  AdoTable.TableName := (BdeTable.TableName);
  // check if rhe table already exists
  while TableExists (AdoTable.TableName) do
    AdoTable.TableName := AdoTable.TableName + 'New';
  Memo1.Lines.Add ('create table ' +
    AdoTable.TableName + ' (');
```

Metod `TableExists` koristi metod `GetTableNames` komponente ADOConnection, koju smo već videli u prethodnom primeru. Kada se formira naziv tabele, program generiše prvu liniju komande, stringom *'create table'*. Zatim je potrebno da dodamo po jednu liniju za svako polje, naznačavajući naziv polja i tip podataka polja (u ovom primeru koristimo MS Access tipove podataka). To se postiže skeniranjem definicija polja:

```
// btnGetStructureClick continued...
// get field information
BdeTable.FieldDefs.Update;
for I := 0 to BdeTable.FieldDefs.Count - 1 do
begin
  strField := ' '
    BdeTable.FieldDefs[I].Name + ' ' +
    AdoTypeName (BdeTable. FieldDefs[I]);
  // add comma or parenthesis
  if I < BdeTable.FieldDefs.Count - 1 then
    strField := strField + ','
  else
    strField := strField + ')';
  Memo1.Lines.Add (strField);
end;
```

Stvarni posao se obavlja funkcijom `AdoTypeName` (preciznije ime bi bilo AccessTypeName). Funkcija proverava tip podataka polja i daje odgovarajuću definiciju. Evo početnog dela funkcije:

```
function AdoTypeName (fdef: TPieldoef): string;
begin
  case fdef.DataTyPe of
    ftString: Result := 'TEXT(' +
      IntToStr (fdef.Size) + ')';
    ftSmallint: Result := 'SMALLINT';
    ftInteger: Result := 'INTEGER';
    ftWord: Result := 'WORD';
    ftBoolean: Result := 'YESNO';
    ...
```

Ovim kodom možemo izraditi SQL iskaze slične onima na slici 12.7. Kod preostale dve kontrole je jednostavniji. Druga kontrola jednostavno preuzima tekst memo kontrole, kada je korisnik po potrebi izmeni, i izvršava kod:

```
procedure TForm1.btnCreateTableClick(Sender: TObject);
begin
  ADOCommand.CommandText := Memo1.Text;
  ADOCommand.Execute;
end;
```

Kada je tabela na raspolaganju i kada ima istu strukturu kao i originalna tabela, poslednja kontrola prebacuje podatke. Pošto ne možemo da koristimo komponentu BatchMove, jednostavno ručno skeniramo tabelu i premeštamo polja jedno po jedno. Ovo svakako nije najbolji pristup, ali nam može pomoći pri prevazilaženju razlika u implementaciji: čak i kada su tipovi polja slični, stvarna struktura slogova u memoriji se može malo razlikovati. Upotrebom naziva polja izvorne tabele kod će funkcionisati čak i kad odredišna tabela ima i neka dodatna polja (ali suprotno ne važi):

```
procedure TForm1.btnMoveDataClick(Sender: TObject);
var
  I: Integer;
begin
  BdeTable.Open;
  AdoTable.Open;
  try
    // for each record
    while not BdeTable.Eof do
    begin
      // new record
      AdoTable.Insert;
      // for each field
      for I := 0 to BdeTable.Fields.Count - 1 do
  with BdeTable.Fields[I] do
          AdoTable.FieldByName (Name).Value := Value;
        // post and move on
        AdoTable.Post;
        BdeTable.Next;
    end;
  finally
    BdeTable.Close;
    AdoTable.Close;
  end;
end;
```

Master/Detail strukture

Komponente ADODataSet i ADOTable sadrže identičnu podršku za master-detail zavisnosti komponente Table. Možete odrediti *master* izvor podataka i povezati nekoliko polja. Da bih dokazao da je ovo zaista moguće, izradio sam jednostavnu master/detail strukturu koristeći tabele kupaca/narudžbina/proizvoda baze podataka DBDEMOS, pošto sam ih konvertovao u Access upotrebom programa Bde2Ado koji sam upravo opisao.

Da biste povezali dve ADOTable komponente, potrebno je da podesite svojstva `MasterFields` i `MasterSource`, kao u sledećem DFM fajlu:

```
object ADOTable2: TADOTable
  Connection = ADOConnection
  IndexFieldNames ='CustNo'
  MasterFields = 'CustNo'
  MasterSource = DataSource1
  TableName = 'orders'
end
```

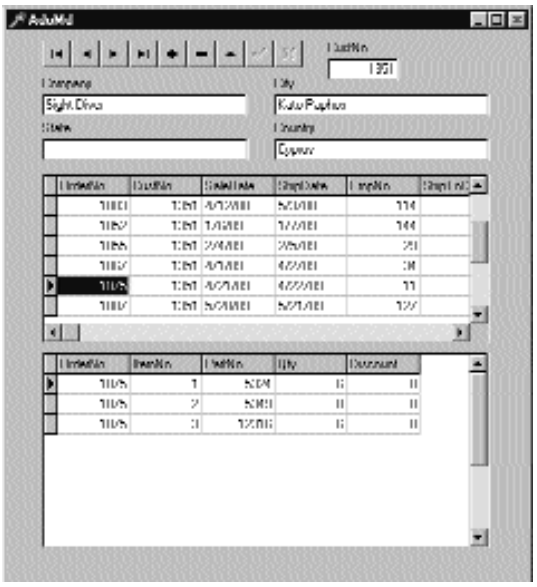
Kada upotrebljavate ADODataSet komponente, koristite svojstva DataSource i MasterFields:

```
object ADODataSet3: TADODataSet
  Connection = ADOConnection
  CommandText = 'items'
  CommandType = cmdTable
  DataSource = DataSource2
  IndexFieldNames = 'OrderNo'
  MasterFields = OrderNo'
end
```

Primetićete da je skup podataka povezan sa tabelom; kada koristite upit, potrebno je da koristite parametre, kao što smo to činili za BDE upite. Izlaz ovog programa možete videti na slici 12.8. Upotrebom modula podataka možete definisati dijagrame, kao kod tradicionalnih master/detail programa, ali sam ja u ovom slučaju izradio program tako što sam samo postavio komponente za pristup podacima na formular.

NAPOMENA

Kada želite da spojite dve tabele, možete napisati upit sa iskazom spajanja, što smo pokazali u prethodnom poglavlju, ili možete upotrebiti master/detail strukturu. Alternativno, možete prepustiti korisniku da kreira tabelu koja sadrži dodatno polje koje odgovara detail tabeli. Ovo je podržano kod nekih objektnih relacionih servera baza podataka (kao što je Oracle 8) i takođe se može koristiti u ADO-u upotrebom tehnike koja se naziva Data Shaping. Data Shaping omogućava da na serveru definišete polja koja se izračunavaju i polja koja se referišu na detail tabele. Na primer, možete upotrebiti sledeću ADO komandu: `SHAPE {select * from customer} APPEND ({select * from orders} AS Orders RELATE CustNo TO CustNo)` da biste umetnuli tabelu orders kao polje tabele customer. Postoji primer ovog pristupa u Ado\Shape direktorijumu Delphi Demos direktorijuma. Delphi MIDAS arhitektura i komponenta ClientDataSet daju sličnu podršku ugnežđenim tabelama. ■



SLIKA 12.8 *Primer AdoMd demonstrira master/detail zavisnost između ADO komponenata*

Još karakteristika ADO-a

Do sada smo se upoznali samo sa osnovama ADO-a. Ipak, ja ne mogu da objasnim sve elemente programiranja baza podataka upotrebom ADO-a jer bi za to bila potrebna posebna knjiga, a samo vlasnici Enterprise izdanja Delphija bi mogli da imaju koristi od takvog razmatranja. Ipak, neke specifične karakteristike su važne. Da bih istakao neke od tih karakteristika, izradio sam nekoliko dodatnih primera.

Kursori i optimizacija

Proces optimizacije baze podataka je veoma specifičan za određene tipove baza podataka (lokalne ili SQL servere) i za drajvere koji se koriste za pristupanje podacima; dakle, ja ne mogu da razmatram svaku karakteristiku optimizacije ADO-a. Ipak, postoje neke karakteristike koje možete testirati kada tražite rešenja koja će poboljšati bazu podataka.

Ključni element koji utiče na brzinu ADO modela jeste pozicija kursora na klijentu ili na serveru, a koja je naznačena svojstvom `CursorLocation` komponente ADODataset. Naravno, ovo je veoma važno za klijent/server operacije:

- Upotreba kursora na strani klijenta znači premeštanje svih podataka do klijenta pre nego što počnete da ih koristite. Ova procedura može inicijalno usporiti operaciju, ali ćete primetiti pristup podacima kada se nađu na klijentu. Takođe, možete izvršiti veliki broj operacija nad lokalnim podacima, kao što je sortiranje, a da ne morate ponovo zahtevati podatke sa servera. Kursor na strani klijenta se ponaša kao lokalni keš.

- Upotreba kursora na strani servera znači dobijanje samo slogova koji su zahtevani, a od servera se traži više kada korisnik pregleda podatke. Ovo aplikaciju može učiniti bržom na početku, ali u nekim slučajevima performanse mogu opasti. Na primer, ponovno sortiranje slogova znači izvršavanje novog zahteva nad bazom podataka. Takođe ćete primetiti da ne možete upotrebiti kursor na strani klijenta ukoliko su podaci lokalni, recimo, kada je u pitanju lokalna Access tabela.

Pozicija kursora je striktno povezana sa tipom kursora, što je naznačeno svojstvom `CursorType`. Tip kursora `ctOpenForwardOnly`, na primer, omogućava samo jednosmerne operacije koje su tipično podržane od strane SQL servera (kao što smo razmatrali u Poglavlju 11). Kod nekih SQL servera pomeranje unazad za jedan slog može značiti ponovno izvršavanje upita i ponovno prihvatanje prethodnih slogova. Ipak, ukoliko unapred znate da ćete pregledati ceo skup slogova u jednom smeru (kao kada, recimo, izrađujete izveštaj na papiru ili za Web), bolje performanse ćete dobiti upotrebom kursora forward-only (jer sistem ne mora da kešira podatke koje neće ponovo koristiti).

Ostali tipovi kursora (`ctKeyset`, `ctDynamic` i `ctStatic`) određuju kako će ažuriranja drugih korisnika uticati na podatke koje čitamo.

- Dinamički (dynamic) kursor je najfleksibilniji tip kursora. Omogućava Vam da vidite slogove koje su dodali, ažurirali ili uklonili drugi korisnici, kao i da u oba smera pretražujete podatke. Takođe, on podržava oznake (bookmarks), ali samo ukoliko ih podržava i provajder podataka.
- *Keyset* kursor funkcioniše slično dinamičkom, ali ne možete videti slogove koje su dodali ili uklonili drugi korisnici.
- Statički (`static`) kursor je najrestriktivniji: daje Vam zamrznutu sliku podataka u vreme Vašeg zahteva i nećete videti izmene podataka koje su načinili drugi korisnici. Ovaj tip kursora se obično koristi za generisanje izveštaja i za proučavanje nepromenljivih skupova podataka.

Na način na koji se dobijaju podaci takođe utiče `ExecuteOptions` (kada možete zahtevati asinhrone operacije, ukoliko ih provajder podataka podržava). Na primer, ukoliko odredite veličinu vrednosti `Cache` i `eoAsyncFetch` za `ExecuteOptions`, ADO odmah pribavlja inicijalnu količinu naznačenu svojstvom `Cache`, a zatim asinhrono dobavlja preostale redove.

Konačno, ukoliko želite da slogove čitate iz memorije da biste sa njima radili, a da ne morate osvežavati interfejs, možete da podesite svojstvo `BlockReadSize` komponente ADODataSet. Kada je vrednost veća od nule, vrednost svojstva `State` će biti `dsBlockRead`, a kontrole koje prepoznaju podatke se neće osvežavati. Prednost je dodatna brzina prilikom dobavljanja više slogova.

Indeksi i sortiranje

Upotrebom kursora na strani klijenta možete kreirati privremene (ili interne) indekse. Na ovaj način se mogu ubrzati operacije pronalaženja, sortiranja i filtriranja. Da biste kreirali privremeni indeks, potrebno je da dodate dinamičko svojstvo `Optimize` odgovarajućem ADO objektu polja.

Šta je dinamičko svojstvo? Dinamička svojstva u ADO-u su opciona svojstva, koja mogu ili ne moraju biti prisutna. U praksi, `Properties` je kolekcija promenljivih vrednosti od kojih svaka ima svoj naziv. Mnogi ADO objekti imaju ovo svojstvo.

Mada Delphi ADO polja ne sadrže obavezno svojstvo `Optimize`, njegovo dodavanje odgovarajućem objektu polja nije tako teško kao što izgleda.

Pošto Delphi ne nudi ovu karakteristiku, verovatno mislite da je njena upotreba veoma složena. Sasvim suprotno, komponenta ADODataSet prikazuje interni `Recordset` ADO objekat koji sadrži svojstvo `Fields`, koje čuva kolekciju ADO polja. Svako polje sadrži kolekciju Properties kojoj možete pristupiti da biste odredili ili pročitali vrednost svakog elementa. U terminima koda to izgleda ovako:

```
AdoDataSet.Recordset.Fields[I].
Properties['Optimize'].Set_Value (True);
```

Ovaj kod koristi program AdoSort za kreiranje privremenog indeksa nad brojem polja I tabele. Kada podesite indeks, mogu ga koristiti svojstva `Sort` i `Filter` i metod `Find`. (Kada tabelu sortirate prema polju, u svakom slučaju se kreira privremeni indeks, čineći manuelnu operciju gotovo beskorisnom. Postoji ipak jedna mala prednost: ukoliko ručno kreirate privremeni indeks, možete biti sigurni da će biti sačuvan i da će ga koristiti više operacija. Ukoliko se oslonite na automatsko indeksiranje, indeks će se možda kreirati svaki put iznova.)

Jasno je da je jedan od ciljeva programa AdoSort sortiranje. ADO DataSet sadrži svojstvo `Sort` u kome možete izlistati polja koja koristite za operaciju sortiranja, verovatno upotrebom kursora na strani klijenta. Ovom svojstvu možete dodeliti naziv polja ili više naziva.

U primeru AdoSort lista se popunjava nazivima polja prilikom kreiranja formulara:

```
procedure TFormSort.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to AdoDataSet.FieldDefs.Count - 1 do
    ListFields.Items.Add (AdoDataSet.FieldDefs [I] Name)
end;
```

Korisnik može odabrati polje i prevući ga u jedno od polja za izmene koja se nalaze ispod, kopirajući u njega tekst:

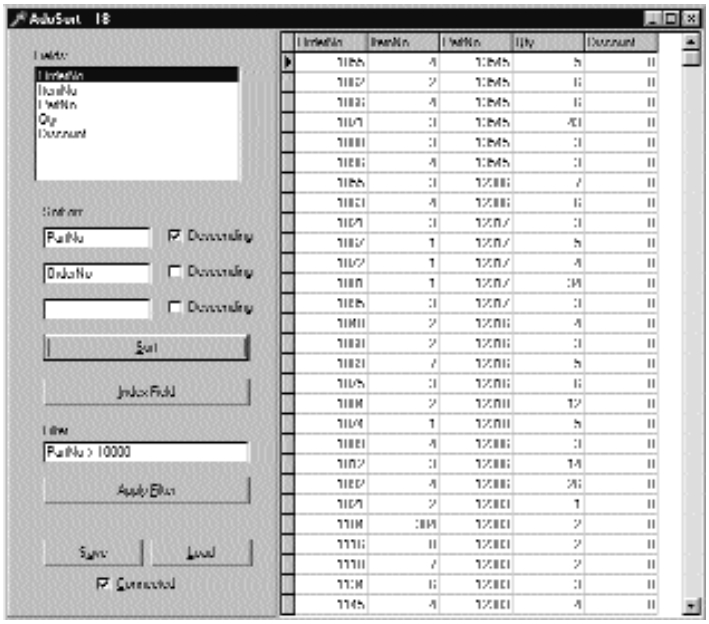
```
procedure TFormSort.Edit1DragDrop(Sender, Source: TObject;
  X, Y: Integer);
begin
  (Sender as TEdit).Text := (Source as TListBox).Items [
    (Source as TListBox).ItemIndex];
end;
```

Polja za izmene se mogu isprazniti kada kliknete na njih.

Kada kliknete kontrolu Sort, sva polja u kojima su prikazani nazivi u poljima za izmene, biće upotrebljena za operaciju sortiranja. (Vreme izvršavanja operacije se meri Windows API funkcijom `GetTickCount`.) Program dodaje ključnu reč `'DESC'` nazivu polja kada je potrebno sortiranje u

opadajućem redosledu (kao na slici 12.9). Kod biste mogli da poboljšate dodavanjem reference, koja je više generička, na komponente, ali i ovako funkcioniše:

```
procedure TFormSort.btnSortClick(Sender: TObject);
var
  t: Cardinal;
  strSort: string;
begin
  t := GetTickCount;
  strSort := Edit1.Text;
  if CheckBox1.Checked then
    strSort := strSort + 'DESC'
  if Edit2.Text <> '' then
    strSort := strSort + ',' + Edit2.Text;
  if CheckBox2.Checked then
    strSort := strSort + 'DESC'
  if Edit3.Text <> '' then
    strSort := strSort + ',' + Edit3.Text:
  if CheckBox3.Checked then
    strSort := strSort + ' DESC';
  AdoDataSet.Sort := strSort;
  Caption := 'AdoSort - ' + IntToStr (GetTickCount - t);
end;
```



SLIKA 12.9 *Izlaz programa AdoSort, sa delimično opadajućim indeksom*

Ukoliko uključite indeks za polje pre sortiranja, videćete ograničenu vremensku razliku u poređenju sa direktnim sortiranjem. Kada jednom obavite operaciju, indeks se čuva u memoriji i razlika će nestati.

Filtriranje

Poslednja karakteristika koju demonstrira program AdoSort jeste filter. Kod ADO Recordset objekta jedno svojstvo `Filter` može da se koristi za tri različite operacije, koje Delphi daje preko tri različita svojstva:

- Svojstvo `Filter` čuva uobičajeni niz uslova, kao što je SQL klauzula Where. Ovo je slično svojstvu `Filter` komponente Table i može da sadrži više uslova odvojenih operatorima `AND` i `OR`.
- Svojstvo `FilterBookmarks` Vam omogućava da prosledite kolekciju oznaka slogovima koje želite da prikažete. Ovo je vrsta selektora slogova.
- Svojstvo `FilterGroup` Vam omogućava da filtrirate slogove prema njihovom statusu. Možete odabrati izmenjene slogove (samo za ažuriranja u pozadini), slogove koji su pogođeni poslednjim operacijama umetanja ili uklanjanja i tako dalje.

Filtriranje se zaista obavlja samo ukoliko odredite vrednost `True` za svojstvo `Filtered`. Primer AdoSort koristi tradicionalniji pristup koji Vam omogućava da tekst polja za izmene prebacite u svojstvo `Filter`.

Trenutni snimak podataka

Kada imate podatke u skupu slogova, možda želite da radite sa njima a da ne morate da budete povezani sa bazom podataka, na primer, kada koristite prenosivi kompjuter. ADO omogućava da to učinite tako što Vam dopušta da skup slogova sačuvate u lokalnom fajlu i da ga zatim ponovo učitate.

NAPOMENA

Ista karakteristika postoji i za Delphijevu komponentu ClientdataSet i naziva se Briefcase model. Komponenta ClientDataSet je opisana u Poglavlju 21. ■

U primeru AdoSort možete koristiti kontrole u dnu panela da biste sačuvali trenutni snimak skupa slogova u fajlu i da ponovo učitate neki od tih fajlova. Kada čuvate podatke, Vi čuvate trenutnu situaciju sa trenutnim sortiranjem i filtriranjem. Na ovaj način možete da sačuvate samo podatke za koje ste zainteresovani.

Da biste sačuvali podatke, pozivate metod `SaveToFile` kojem je kao parametar potreban naziv fajla. Neke od njegovih karakteristika nisu intuitivne: ne možete upotrebiti postojeći fajl, fajl ostaje otvoren i ažurira se sve dok ne zatvorite skup podataka (predstavlja nešto kao stalni keš) i potrebno je da koristite kursor na strani klijenta ukoliko provajder ne podržava ovu operaciju. Evo koda:

```
procedure TFormSort.btnSaveClick (Sender: TObject);
begin
  if SaveDialog.Execute and not
      FileExists (SaveDialog.FileName) then
    AdoDataSet.SaveToFile (SaveDialog.FileName);
end;
```

Ukoliko želite da fajl učinite dostupnim, potrebno je prvo da zatvorite skup podataka koristeći polje za potvrdu koje se nalazi na dnu:

```
procedure TFormSort.cbConnectedClick (Sender: TObject);
begin
  AdoDataSet.Active := cbConnected.Checked;
end;
```

Kada ovu operaciju ponovite više puta, možete da zatvorite skup podataka i ponovo učitate jedan od snimaka koji se nalazi u fajlu:

```
procedure TFormSort.btnLoadClick (Sender: TObject);
begin
  if OpenDialog.Execute then
    AdoDataSet.LoadFromFile (OpenDialog.FileName);
  cbConnected.Checked := True;
end;
```

Pronalaženje, sumiranje i zaključavanje slogova

Da biste pronašli slog koji se nalazi u trenutnom skupu slogova, možete upotrebiti standardni metod Locate. Kao i kod BDE skupa podataka, možete potražiti vrednosti jednog ili više polja koja se prosleđuju nizom. U primeru AdoEmpl (koji se zasniva na tabeli Employee koja je prebačena u Access iz DBDEMS-a) ovaj kod je pridružen kontroli Find:

```
procedure TAdoEmplForm.btnFindClick (Sender: TObject);
begin
  if not AdoTable.Locate (
      'LastName', EditName.Text, []) then
    MessageDlg ('Name not found', mtError, [mbOK], 0);
end;
```

Još jedna karakteristika, koja se koristila u primeru Total, Poglavlja 9, jeste mogućnost izračunavanja sume plata zaposlenih. Umesto da to učinimo upotrebom specifičnog SQL upita, program skenira tabelu, što smo do sada videli više puta. U prethodnim primerima smo aktivirali i deaktivirali korisnički interfejs kodom nalik na ovaj:

```
AdoTable.DisableControls;
try
  //code
finally
  AdoTable.EnableControls;
```

Ovoga puta možemo da upotrebimo tehniku čitanja bloka, koja onemogućava vizuelne kontrole i trebalo bi da poboljša performanse:

```
AdoTable.BlockReadSize := 10;
try
  // code
finally
  AdoTable.BlockReadSize := 0;
```

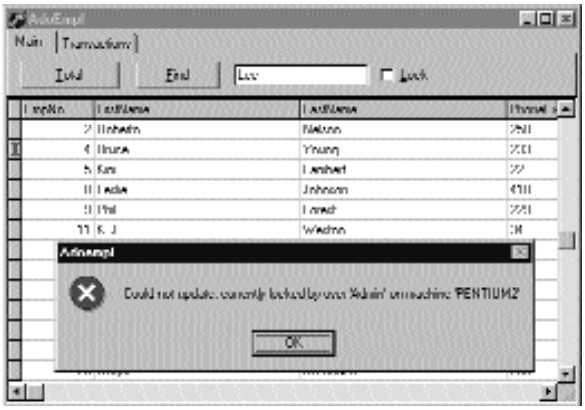
Program Vam, takođe, omogućava da eksperimentišete tipovima zaključavanja, mada je MS Access podrška ograničena u poređenju sa ADO-om. U prošlom poglavlju smo videli da Paradox zaključava slog prilikom editovanja. ADO Vam omogućava da naznačite strategiju zaključavanja upotrebom svojstva `LockType` skupa podataka. Možete da odaberete pesimističko zaključavanje,

kada dva korisnika ne mogu istovremeno da edituju isti slog, i optimističko zaključavanje, kada svaki korisnik može da zameni bilo čije izmene.

Kada kliknete polje za potvrdu Lock, menja se status svojstva `LockType`:

```
procedure TAdoEmplForm.cbLockClick (Sender: TObject);
begin
  AdoTable.Click;
  if not cbLock.Checked then
    AdoTable.LockType := ltPessimistic
  else
    AdoTable.LockType := ltOptimistic;
  AdoTable.Open;
end;
```

Aktiviranjem zaključavanja (pa čak i kada ga ne aktivirate!) i editovanjem istog sloga od strane dveju instanci programa, trebalo bi da vidite poruku o grešci, koja je prikazana na slici 12.10. Primetićete da efekat koji dobijate izvršavanjem programa zavisi od toga da li program koristi transakcije: kada isključite transakcije, dobićete drugačiju poruku.



SLIKA 12.10 *Poruka o grešci do koje dovodi zaključavanje u primeru AdoEmpl*

UPOZORENJE

Imajte na umu da će Access prijaviti grešku samo kada se pošalje slog, a ne kada započne operacija editovanja. Ovo svakako nije dobro za korisnika. Takođe, menjanje sloga zaključava taj i nekoliko susednih slogova jer Access koristi strategiju zaključavanja strane. Dakle, ukoliko korisnik menja slog, drugi korisnik neće moći da promeni isti slog, ali ni nekoliko slogova pre i posle sloga koji se menja. ■

Rukovanje transakcijama u ADO-u

Kao i BDE, ADO Vam omogućava da rukujete transakcijama. Da bih pokazao kako to ADO obavlja, ja sam preuzeo kontrole primera Transact i njihov kod, i dodao sam ih primeru AdoEmpl, na drugoj strani Page Control koji je upotrebljen za izradu palete alata koja sadrži dve strane.

Ovaj kod je veoma sličan ranijem BDE primeru, a promenjen je samo naziv metoda. (Prilično čudna situacija, moram da priznam.) Metodi `StartTransaction`, `Commit` i `Rollback` komponente Database postali su metodi `BeginTrans`, `CommitTrans` i `RollbackTrans` komponente AdoConnection.

Da bi se uklonio kod koji se koristio za aktiviranje i deaktiviranje kontrola, evo srži transakcije koja obrađuje podršku u primeru AdoEmpl:

```
procedure TAdoEmplForm.BtnCommitClick (Sender: TObject);
begin
  if AdoTable.State := dsEdit then
    AdoTable.Post;
  AdoConnection.CommitTrans;
end;

procedure TAdoEmplForm.BtnRollbackClick (Sender: TObject);
begin
  AdoTable.Cancel;
  AdoConnection.RollbackTrans;
  // refresh
  AdoTable.Requery;
end;

procedure TAdoEmplForm.BtnStartClick (Sender: TObject);
begin
  AdoConnection.BeginTrans;
end;
```

U prethodnom poglavlju smo videli da komponenta Database omogućava izbor izolacije transakcije (*izlolacija* između transakcija korisnika u višekorisničkom okruženju) upotrebom svojstva `TransIsolation`. U ADO-u možete da kontrolišete interakciju između transakcija podešavanjem svojstva `IsolationLevel` komponente ADOCOnnection pre otvaranja veze. Tri važne opcije su:

- `ilReadCommitted` (unapred određena vrednost) i ekvivalent znače da možemo videti samo upisane izmene koje su načinili drugi korisnici. Ovo je obično željeni nivo izolacije.
- `ilReadUncommitted` i ekvivalent `ilBrowse` znače da možemo videti izmene koje su načinili drugi korisnici čak i kada nisu upisane. Kako drugi korisnici mogu da promene mišljenje i opozovu transakciju, ne možemo se osloniti na validnost podataka sa kojima radimo, što ovo podešavanje ograničava na nekoliko slučajeva.
- `ilRepeatableRead` znači da nećete videti izmene koje su načinile druge transakcije, čak i kada su izmene upisane, izuzev ukoliko ponovo ne izvršite upit. Ovo nije moguće kada je podešena vrednost `ilIsolated`, što predstavlja najrestriktivniji nivo izolacije.

Iako ADO omogućava veliki broj podešavanja za nivo izolacije transakcije, većina provajdera (baze podataka) podržava manje opcija. Može se desiti da na raspolaganju imate manje nivoa izolacije nego što ste zahtevali.

Korisnički događaji

Pored uobičajenih događaja skupa podataka, postoji nekoliko interesantnih događaja ADO komponenata. Delphi skup podataka sadrži događaje *Before* i *After*; ADO sadrži događaje *Will* i *Complete*.

Na primer, komponenta za povezivanje sadrži događaje `OnWillConnect` i `OnConnectComplete` i događaje `OnWillExecute` i `OnExecuteComplete`. Možete upotrebiti `OnWillExecute` da zabeležite svaku komandu koja je poslata serveru, ili da izdvojite komande koje nisu autorizovane za korisnike (kao što je komanda delete). Ova komponenta sadrži mnoge događaje koji se odnose na transakcije.

Komponenta ADODataSet sadrži događaje *Will* i *Complete* za izmenu polja, izmenu sloga, izmenu skupa slogova i za operacije premeštanja. Takođe, sadrži događaje `OnFetchProgress` i `OnFetchComplete` koje možete upotrebiti za implementiranje ProgressBara dok se čeka na učitavanje velikog skupa podataka.

Šta je sledeće?

U ovom poglavlju sam predstavio ključne ideje koje se odnose na upotrebu Microsoft ADO interfejsa i OLE DB infrastrukture za programiranje baza podataka. Kako Microsoft integriše ove komponente u operativni sistem, možete očekivati da ADO postane popularniji.

Borlandova odluka da podržava ADO osnovnim DataSet komponentama omogućava Vam da postojeće programe lako konvertujete i adaptirate na novu tehnologiju (ADO) dok još uvek možete da koristite arhitekturu koju obezbeđuje `TDataSet` klasa VCL-a. Mada ovo nije jedini mogući način za rad sa ADO-om, sigurno je dobar izbor.

Druga dobra stvar je da ukoliko sledite ovaj pristup, možete se rešiti BDE-a na klijent kompjuterima, i da instalacija na drugim klijent kompjuterima, verovatno, bude manja. Za blisku budućnost pretpostavite da ćete morati da instalirate ADO na starije operativne sisteme koji ga još nemaju. ADO čak podržava pristup udaljenim izvorima podataka u 3-tier arhitekturi. Prema onome što sam ja video, ova arhitektura je ograničenija od Borlandove MIDAS tehnologije.

Ovo poglavlje završava deo knjige koji je posvećen programiranju baza podataka, ali ću se na ovu temu ponovo vratiti kada budem razmatrao višelinijske upite u Poglavlju 17, kada budem predstavljao upotrebu baza podataka u Internet programiranju u Poglavlju 20, i kada budem razmatrao MIDAS u Poglavlju 21.

DEO

IV

Komponente i biblioteke

U OVOM DELU:

POGLAVLJE

13

Kreiranje komponenata

Dok je verovatno većina Delphi programera vešta u upotrebi postojećih komponenata, ponekad je ipak korisno napisati sopstvene komponente ili prilagoditi postojeće. Jedan od najinteresantnijih aspekata Delphija je taj da je kreiranje komponenata jednostavno. Zbog toga, mada je ova knjiga namenjena Delphi programerima a ne onima koji u Delphiju pišu komponente, ovo poglavlje će se pozabaviti kreiranjem komponenata i predstavljanjem Delphi dodataka (add-in), kao što su komponente i editori svojstava.

Ovo poglavlje daje pregled Delphi komponenata i predstavalja veliki broj primera. Nema dovoljno mesta da se predstave veoma složene komponente, ali ideje sa kojima ćete se upoznati u ovom poglavlju, odnose se na sve osnovne stvari, što Vam je dovoljno da počnete da pišete svoje komponente.

NAPOMENA

Mnogo više informacija ćete pronaći u knjizi "Delphi Developer's Handbook" (Sybex, 1998), među kojima i uputstvo kako da izradite komponente koje prepoznaju podatke i mnoge druge napredne tehnike. ■

Proširivanje VCL-a

Kada pišete novu komponentu, uvek proširujete jednu od postojećih klasa VCL-a. Da biste to učinili, možete da upotrebite mnoge karakteristike jezika Object Pascal, koje su korisnicima komponenata retko potrebne. Ukoliko još uvek imate nedoumice oko naprednih kakrakteristika Object Pascala, mogli biste da pregledate celoukupni opis jezika koji se nalazi u prvom delu ove knjige. Poglavlje 3 daje pregled uloge svojstava, metoda i događaja, a Poglavlje 4 predstavlja uvod u strukturu VCL-a. Ukoliko ste preskočili ova poglavlja ili se ne osećate sigurno kada su u pitanju osnovne ideje VCL-a, pročitajte ova poglavlja pre nego što nastavite dalje.

Delphi komponente su klase, a VCL je kolekcija svih klasa koje definišu Delphi komponente. Svaki put kada Delphiju dodate novi paket sa nekim komponentama, Vi, zapravo, proširujete VCL novom klasom. Ta nova klasa će biti izvedena iz jedne od postojećih klasa koje se odnose na komponente, dodajući nove mogućnosti klasama koje će iz nje biti izvedene.

Novu komponentu možete izvesti iz postojeće komponente ili iz apstraktne klase komponente (abstract component class) — klase koja ne odgovara upotrebnim komponentama. VCL hijerarhija sadrži mnoge od ovih međuklasa da bi Vam omogućila da odaberete unpared određeno ponašanje za nove komponente i promenite njihova svojstva.

Paketi komponenata

Komponente se dodaju u pakete komponenata. Svaki paket komponenata je u osnovi DLL (dinamička biblioteka za povezivanje) sa ekstenzijom BPL (što je skraćenica za Borland Package Library).

Paketi postoje u dva oblika: paketi u vreme dizajniranja, koje koristi Delphi IDE, i paketi u vreme izvršavanja, koje po potrebi koriste aplikacije. Opcija za vreme dizajniranja ili za vreme izvršavanja određuje tip paketa. Kada pokušate da instalirate paket, IDE proverava da li paket ima zastavice samo za dizajniranje ili samo za izvršavanje, i odlučuje da li da korisniku dozvoli instaliranje paketa i da li da paket doda listi paketa samo za izvršavanje. Pošto postoje dve opcije koje se međusobno ne isključuju, a svaka ima dva moguća stanja, postoje četiri različite vrste paketa komponenata — dve glavne varijacije i dva specijalna slučaja:

- Komponente samo za vreme dizajniranja se mogu instalirati u Delphi okruženje. Ovi paketi obično sadrže delove komponenata koje se koriste u vreme dizajniranja, kao što su editori svojstva i kod za registraciju. Često mogu da sadrže i same komponente, mada to nije najprofesionalniji pristup. Kod komponenata paketa u

vreme dizajniranja, obično se statički povezuje u izvršni fajl, upotrebom koda odgovarajućih Delphi Compiled Unit (DCU) fajlova. Imajte na umu da ove pakete možete koristiti i kao pakete za vreme izvršavanja.

- Komponente samo za vreme izvršavanja koriste Delphi aplikacije u vreme izvršavanja. Ovakvi paketi se ne mogu instalirati u Delphi okruženje, ali se automatski dodaju listi paketa samo za vreme izvršavanja kada budu bili potrebni paketi samo za vreme dizajniranja koje instalirate. Ovi paketi obično sadrže kod klasa komponente, ali nema podrške za vreme dizajniranja (to je da bi se maksimalno smanjila veličina biblioteka komponenata koja se prosleđuje uz Vaše izvršne fajlove). Paketi samo za vreme izvršavanja su važni jer se mogu slobodno distribuirati uz aplikacije, ali drugi Delphi programeri neće moći da ih instaliraju u okruženje da bi mogli da ih koriste u novim programima.
- Obični paketi komponenata (koji nemaju određenu opciju samo za dizajniranje ili samo za izvršavanje) ne mogu se instalirati i neće automatski biti dodati listi paketa samo za vreme izvršavanja. Ovo ima smisla kada su u pitanju pomoćni paketi koje koriste drugi paketi, ali ovakav tip paketa se svakako retko sreće.
- Paketi kojima su određene obe zastavice, mogu se instalirati i automatski se dodaju listi paketa samo za vreme izvršavanja. Ovakvi paketi obično sadrže komponente kojima je potrebna mala ili nikakva podrška u vreme dizajniranja (na stranu ograničeni registracioni kod komponente). Imajte na umu da korisnici aplikacija koje su izrađene ovakvim paketima mogu da koriste pakete u svom programiranju.

SAVET

Nazivi fajlova Delphijevih paketa samo za vreme dizajniranja započinju slovima DCL (na primer, DCLSTD50.BPL); nazivi fajlova paketa samo za vreme izvršavanja započinju slovima VCL (na primer, VCL50.BPL). Istu notaciju možete upotrebiti za svoje pakete, ukoliko to želite. ■

U Poglavlju 1 smo razmatrali efekat paketa na veličinu izvršnog fajla programa. Sada ćemo obratiti pažnju na izradu paketa, jer je to neophodan korak u kreiranju ili instaliranju komponenata u Delphiju.

Kada kompajlirate paket samo za vreme izvršavanja, Vi proizvodite dinamičku biblioteku za povezivanje sa kompajliranim kodom (BPL fajl) i fajl sa simboličkim informacijama (DCP fajl), koji sadrži nekompajlirani mašinski kod. Ovaj drugi fajl koristi Delphi kompajler za prikupljanje simboličkih informacija o jedinicama koje su deo paketa, a da ne mora da pristupa fajlovima jedinica (DCU) koji sadrže simboličke informacije i kompajlirani mašinski kod. Ovim se smanjuje vreme potrebno za kompajliranje i omogućava Vam se da distribuirate samo pakete bez unapred kompajliranih fajlova jedinica. Unapred kompajlirane jedinice su još uvek potrebne za statičko povezivanje komponenata u aplikacije. Distribucija unapred kompajliranih DCU fajlova (ili izvornog koda) ima smisla u zavisnosti od tipa komponenata koje pišete. Kreiranjem komponenata ćemo se pozabaviti posle razmatranja nekih opštih smernica i pošto izradimo našu prvu komponentu.

NAPOMENA

DLL-ovi su izvršni fajlovi koji sadrže kolekcije funkcija i klasa, koje se mogu koristiti u aplikacijama ili drugim DLL-ovima u vreme izvršavanja. Tipična prednost je u tome da je, ukoliko mnoge aplikacije koriste isti DLL, potrebno da postoji samo jedna kopija na disku ili da samo jedna kopija bude učitana u memoriju, a veličina svakog izvršnog fajla će biti mnogo manja. To je ono što se dešava i kada su u pitanju Delphi paketi. Poglavlje 14 se detaljnije bavi DLL-ovima i paketima. ■

Pravila za pisanje komponenata

Neka opšta pravila upravljaju pisanjem komponenata. Detaljan opis većine pravila možete pronaći u Delphi Component Writer's Guide Help fajlu, koji obavezno moraju da pročitaju programeri koji pišu Delphi komponente.

Ovde prikazujem svoj rezime pravila za programere koji pišu komponente:

- Pažljivo proučite jezik Object Pascal. Naročito važni koncepti su nasleđivanje, zaobilaženje metoda i preopterećenje, razlika između sekcija klasa public i published, i definicija svojstava i događaja.
- Pažljivo proučite strukturu VCL hijerarhije klasa i neka Vam pri ruci bude grafički prikaz te hijerarhije (grafički prikaz kakav imate i u Delphiju).
- Poštujte standardne Delphi konvencije imenovanja. Postoji nekoliko takvih konvencija za komponente, što ćemo videti, i poštovanje ovih pravila čini drugim programerima interkaciju sa Vašim komponentama lakšom i omogućava njihovo dalje proširivanje.
- Neka komponente budu jednostavne, neka oponašaju druge komponente i izbegavajte zavisnosti komponenata. Ova pravila u osnovi znače da programeri koji koriste Vaše komponente mogu da ih koriste jednako lako kao komponente koje se instaliraju uz Delphi. Koristite slične nazive za svojstva, metode i događaje kada god je to moguće. Ukoliko korisnici ne moraju da nauče složena pravila o upotrebi Vaših komponenata (to jest, ukoliko su zavisnosti između metoda i svojstava ograničena) i ukoliko jednostavno mogu pristupiti tim svojstvima, biće veoma srećni.
- Koristite izuzetke. Kada nešto pođe naopako, komponente bi trebalo da se pozovu na izuzetak. Kada alocirate resurse bilo koje vrste, morate ih zaštititi `try-finnaly` blokovima i pozivima destruktora.
- Da biste dovršili komponentu, dodajte joj bitmapu koja će se koristiti u Delphi Component Palette. Ukoliko nameravate da Vašu komponentu koristi više programera, razmislite i o dodavanju Help fajla za komponentu.
- Budite spremni da napišete *pravi* kod i zaboravite na vizuelne aspekte Delphija. Pisanje komponenata u osnovi znači pisanje koda bez vizuelne podrške (mada Class Completion može prilično ubrzati kodiranje obične klase). Izuzetak od ovog pravila, u Delphiju 5, je taj da možete da koristite okvire za vizuelno pisanje komponenata.

- Koristite alate nezavisnih programera za pisanje komponenata prilikom izrade Vaših komponenata, ili da biste ubrzali razvoj komponenata. Najmoćniji alat za kreiranje Delphi komponenata, za koji ja znam, jeste Component Development Kit (CDL) koji dolazi iz Eagle Softwarea (`http://www.eagle-software.com`).

Osnovne klase komponenata

Da biste izradili novu komponentu, obično ćete početi od postojeće komponente, ili neke od osnovnih klasa VCL-a. U oba slučaja Vaša komponenta će biti u jednoj od tri kategorije komponenata (koje smo predstavili u Poglavlju 4), određena jednom od tri osnovne klase hijerarhije komponenata:

- `TWinControl` je roditeljska klasa bilo koje komponente koja je bazirana na prozoru. Komponente koje su izvedene iz ove klase mogu dobiti ulazni fokus i mogu primati Windows poruke os sistema. Takođe, možete koristiti njhov hendl prozora prilikom poziva API funkcija. Kada kreirate potpuno novu kontrolu, Vi ćete, uopšte uzev, naslediti iz izvedene klase `TCustomControl`, koja sadrži nekoliko dodatnih korisnih kakrakteristika (naročito podršku za iscrtavanje kontrola).
- `TGraphicControl` je roditeljska klasa vidljivih komponenata koje nemaju Windows hendl (što smanjuje upotrebu nekih Windows resursa). Ove komponente ne mogu da dobiju ulazni fokus ili direktno odgovore na Windows poruke. Kada kreirate potpuno novu grafičku kontrolu, direktno ćete nasleđivati iz ove klase (koja sadrži skup karakteristika koje su veoma nalik na `TCustomControl`).
- `TComponent` je roditeljska klasa svih komponenata (uključujući kontrole) i može se koristiti kao direktna roditeljska klasa za nevizuelne komponente.

U preostalom delu poglavlja izradićemo neke komponente koristeći različite roditeljske klase i videćemo razlike među njima. Započećemo sa komponentama koje nasleđuju od postojećih komponenata ili klasa na niskom nivou hijerarhije, a zatim ćemo videti primere klasa koje nasleđuju direktno od klasa koje smo prethodno pomenuli.

NAPOMENA

Kada izvedete novu komponentu iz ovih klasa visokog nivoa VCL hijerarhije, komponenta već nasleđuje neka svojstva koja su zajednička za sve komponente. Pogledajte ponovo sliku 4.4 da biste videli svojstva koja su definisana u nekim VCL klasama visokog nivoa. ■

Izrada Vaše prve komponente

Kao što smo već videli u Poglavlju 3, veoma je jednostavno uzeti postojeću klasu, pretvoriti je u nevizuelnu komponentu i izraditi jednostavan paket koji će je sadržati. Izrada komponenata je, u stvari, važna aktivnost za Delphi programere. Osnovna ideja je da svaki put kada Vam je potrebno isto ponašanje na dva različita mesta u aplikaciji, ili u dve različite aplikacije, možete smestiti zajednički kod u klasu ili, što je još bolje, u komponentu.

U ovom odeljku ću samo predstaviti nekoliko jednostavnih komponenata da bih Vam dao ideju o koracima koji su potrebni za izradu jedne komponente i da bih Vam pokazao različite stvari koje možete učiniti da biste prilagodili postojeću komponentu sa ograničenom količinom koda.

Combo polje za fontove

Mnoge aplikacije sadrže paletu alata na kojoj se nalazi combo polje iz koga možete da odaberete font. Ukoliko često koristite prilagođeno combo polje, kao što je ovo, zašto ga ne biste pretvorili u komponentu? Za taj postupak bi bilo potrebno verovatno manje od minuta. Da bismo počeli, zatvorite bilo koji aktivni projekat u Delphi okruženju i pokrenite Component Wizard izborom Component➥New Component ili izborom File➥New da biste otvorili Object Respository i zatim odabrali Component na strani New. Kao što možete videti na slici 13.1, Component Wizardu su potrebne sledeće informacije:

- Naziv tipa prethodnika: klase komponente iz koje želite da nasledite. U ovom slučaju mi ćemo upotrebiti `TComboBox`.
- Naziv klase nove komponente koju izrađujete; možemo upotrebiti `TMdFontCombo`.
- Strana Component Palette na kojoj želite da prikažete novu komponentu, što može biti nova ili postojeća strana. Možemo kreirati novu komponentu i nazvati je *Md*.
- Naziv fajla Pascal jedinice u koji će Delphi smestiti izvorni kod nove komponente; možemo uneti `MdFontBox`.
- Trenutna putanja za pretraživanje (koja će automatski biti određena).

SLIKA 13.1 *Definisanje nove komponente TMdFontCombo upotrebom Component Wizarda*

Kliknite kontrolu OK i Component Wizard će generisati sledeći jednostavan Pascal izvorni fajl u kojem će biti struktura Vaše komponente. Kontrola Install može da se koristi za momentalno instaliranje komponente u paket. Pogledajmo prvo kod, a zatim razmotrimo instaliranje:

```
unit FontBox;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics
  Controls, Forms, Dialogs, StdCtrls;
type
  TMdFontCombo = class (TComboBox)
private
  { Private declarations }

protected
  { Protected declarations }

public
  { Public declarations }

published
  { Published declarations }
end;

procedure Register;

implementation

procedure Register;
begin
  RegisterComponents( 'Md', [TMdFontCombo]);
end;

end.
```

Jedan od ključnih elemenata ovog listinga je definicija klase koja započinje naznačavanjem roditeljske klase. Jedini drugi bitan deo je procedura `Register`. Zapravo, videćete da Component Wizard obavlja veoma malo posla.

UPOZORENJE

Počev od Delphija 4, procedura `Register` mora da se piše velikim početnim slovom R. Ovaj zahtev je zbog kompatibilnosti sa C++ Builderom (identifikatori u jeziku C++ razlikuju velika i mala slova). ■

SAVET

Koristite konvencije imenovanja prilikom izrade komponenata. Sve komponente koje su instalirane u Delphiju bi trebalo da imaju različita imena klasa. Zbog toga je većina programera Delphi komponenata odlučila da doda prefiks od dva ili tri slova nazivima njihovih komponenata. I ja sam isto učinio koristeći Md da bih identifikovao komponente koje su izrađene u ovoj knjizi. Prednost ovog pristupa je u tome da možete da instalirate komponentu `TMdFontCombo` čak i kada ste instalirali komponentu imena `TFontCombo`. Imajući na umu da i nazivi jedinica takođe moraju da budu jedinstveni za sve komponente koje su instalirane na sistemu, ja sam dodao isti prefiks nazivima jedinica. Da biste proverili prefikse koje koriste najvažniji dobavljači komponenata, posetite web sajt `http://developers.href.com/dpr`. ■

To je sve što je potrebno za izradu komponente. Naravno, u ovom primeru nema mnogo koda. Potrebna nam je samo kopija svih sistemskih fontova u svojstvu `Items` combo polja prilikom pokretanja. Da bismo ovo postigli, možemo pokušati da zaobiđemo metod `Create` u deklaraciji klase dodajući iskaz `Items := Screen.Fonts`. Ipak, ovo nije korektan pristup. Problem je što ne možemo da pristupimo svojstvu `Items` combo polja pre nego što bude dostupan hendl prozora komponente; komponenta ne može imati hendl prozora sve dok se ne odredi svojstvo `Parent`, a svojstvo `Parent` nije određeno u skupu konstruktora, već kasnije.

Zbog toga, umesto dodeljivanja novih stringova u konstruktoru `Create`, ovu operaciju moramo izvršiti u proceduri `CreateWnd`, koja se poziva za kreiranje prozora kontrole posle konstruisanja komponente, određivanja svojstva `Parent` i kada je dostupan hendl prozora. Još jednom, mi izvršavamo unapred određeno ponašanje, a zatim pišemo naš kod. Mogao sam da preskočim konstruktor `Create` i da napišem sav kod u proceduri `CreateWnd`, ali sam odlučio da koristim oba metoda za pokretanje da bih pokazao razlike koje postoje među njima. Evo deklaracije klase komponente:

```
type
  TMdFontCombo = class (TComboBox)
  public
    constructor Create (AOwner: TComponent); override;
    procedure CreateWnd; override;
  published
    property Style default caDropDownList;
    property Items stored False;
  end;
```

Evo i izvornog koda dva metoda:

```
constructor TMdFontCombo.Create (AOwner: TComponent);
begin
  inherited Create (AOwner);
  Style := csDropDownList;
end;

procedure TMdFontCombo CreateWnd;
begin
  inherited CreateWnd;
  Items.Assign (Screen.Fonts);
end;
```

Primetićete da sam pored dodeljivanja nove vrednosti svojstvu `Style` ove komponente, u metodu `Create`, ponovo definisao ovo svojstvo određivanjem vrednosti ključnom reči `default`. Moramo da obavimo obe operacije jer dodavanje ključne reči `default` deklaraciji svojstva nema direktan efekat na početnu vrednost svojstva. Zbog čega navoditi unapred određenu vrednost svojstva? Zato što se svojstva koja imaju vrednost jednaku unapred definisanoj vrednosti ne uvode u liniju sa definicijom formulara (i ne pojavljuju se u tekstualnom opisu formulara, DFM fajlu). Ključna reč `default` govori kodu da će inicijalizacioni kod komponente odrediti vrednost za to svojstvo.

SAVET

Zbog čega je važno navesti unapred određenu vrednost za svojstvo published? Da bi se smanjila veličina DFM fajlova i, konačno, veličina izvršnih fajlova (koji sadrže DFM fajlove). ■

Drugo ponovo definisano svojstvo, svojstvo `Items`, određeno je kao svojstvo koje ne treba sačuvati u DFM fajlu, bez obzira na aktuelnu vrednost. To se postiže direktivom `stored` za kojom sledi vrednost `False`. Komponenta i njen prozor će biti kreirani kada se pokrene program, pa nema smisla sačuvati je u DFM fajlu koji će kasnije biti odbačen (da bi bio zamenjen novom listom fontova).

NAPOMENA

Mogli smo čak i da napišemo kod metoda CreateWnd da bismo kopirali fontove u elemente combo polja u vreme izvršavanja. To se može postići iskazima kakav je sledeći:

```
if not (csDescending in ComponentState) then
```

Ali, za prvu komponentu koju izrađujemo, manje efikasan ali direktniji metod koji je prethodno opisan jeste jasniji opis osnovne **procedure**. ■

Kreiranje paketa

Sada je potrebno da instaliramo komponentu u okruženje upotrebom paketa. Za ovaj primer možemo kreirati novi paket ili upotrebiti postojeći paket (kao što je unapred određeni "korisnički paket"), kao što smo učinili u Poglavlju 3. Kreiranje novog paketa je, ipak, veoma jednostavno.

U svakom slučaju odaberite komandu menija Component➥Install Component. Rezultujući okvir za dijalog sadrži stranu za instaliranje komponenata u postojeći paket i stranu za kreiranje novog paketa. Kada kliknete OK, otvoriće se Package Editor (videti sliku 13.2) koji se sastoji iz dva dela:

- Lista Contains naznačava komponente koje se nalaze u paketu (ili, da budem precizniji, jedinice koje definišu te komponente).
- Lista Requires naznačava pakete koji su potrebni ovom paketu. Vašem paketu će u opštem slučaju biti potreban paket vcl50 (glavni paket samo za vreme izvršavanja koji sadrži osnovne delove Delphi VCL-a), ali će mu, možda, biti potreban i paket vcldb50 (koji sadrži većinu klasa koje se odnose na baze podataka) ukoliko komponente novog paketa izvršavaju bilo koju operaciju nad bazom podataka.

SLIKA 13.2 *Package Editor*

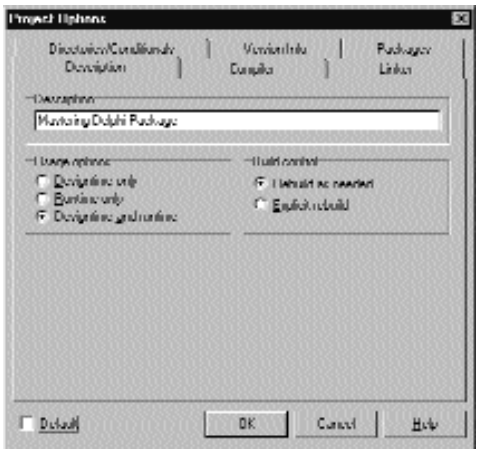
Ukoliko dodate komponentu novom paketu koji smo upravo definisali, a zatim jednostavno kompajlirate paket i instalirate ga (upotrebljavajući dve odgovarajuće kontrole palete alata editora paketa), odmah ćete videti da se nova komponenta prikazuje na Md strani Component Palette. Procedura `Register` fajla jedinice komponente je Delphiju naznačila fde da instalira novu komponentu. Po definiciji, bitmapa koja se koristi biće jednaka bitmapi roditeljske klase, jer nismo obezbedili našu bitmapu (to ćemo učiniti u narednim primerima). Primetićete, takođe, da ukoliko pomerite pokazivač miša iznad nove komponente, Delphi će u oblačiću prikazati naziv klase bez početnog slova T.

Šta se nalazi iza paketa?

Šta se nalazi iza paketa koji smo upravo izradili? Package Editor u osnovi generiše izvorni kod za projekat paketa: specijalnu vrstu DLL-ova ugrađenih u Delphi. Projekat paketa se čuva u fajlu DPK (Delphi PacKage) ekstenzijom. Tipični projekat paketa izgleda ovako:

```
package MdPack;

{$R *.RES}
{$ALIGN ON}
{$BOOLEVAL OF}
{$DEBUGINFO ON}
...
{$DESCRIPTION 'Mastering Delphi Package'}
{$IMPLICITBUILD ON}

requires
  vcl50;

contains
  MdFontBox in 'MdFontBox.pas';

end.
```

Kao što možete videti, Delphi koristi specifične ključne reči za pakete: prva ključna reč je `package` (koja je slična ključnoj reči `library`, koju ću razmatrati u Poglavlju 14). Ova ključna reč uvodi novi projekat paketa. Zatim dolazi lista svih opcija kompajlera, od kojih sam ja neke izostavio iz listinga. Obično se opcije za Delphi projekat čuvaju u zasebnom fajlu; paketi, nasuprot tome, sadrže sve opcije kompajlera direktno u svom izvornom kodu. Među opcijama kompajlera je i direktiva kompajlera `DESCRIPTION`, koja se koristi da bi se opis paketa učinio dostupnim Delphi okruženju. Zapravo, pošto ste instalirali novi paket, njegov opis će se prikazati na strani Packages okvira za dijalog Project Options, strani koju takođe možete aktivirati selektovanjem elementa menija Component➥Install Packages. Ovaj okvir za dijalog je prikazan na slici 13.3.



SLIKA 13.3 *Project Options za pakete. Možete videti novi paket koji smo upravo kreirali.*

Pored uobičajenih direktiva kakva je `DESCRIPTION`, postoje i druge direktive koje su karakteristične za pakete. Uobičajene opcije su lako dostupne upotrebom kontrole Options Package Editora. Posle ove liste opcija dolaze ključne reči `requires` i `contains`, koje daju listu elemenata koji se vizuelno prikazuju na dvema stranama Package Editora. Još jednom, prva je lista paketa koja je neophodna za aktuelni paket, a druga je lista jedinica koju paket instalira.

Kakav je tehnički efekat izrade paketa? Pored DPK fajla sa izvornim kodom, Delphi generiše BPL fajl sa verzijom za dinamičko povezivanje paketa i DCP fajl sa simboličkim informacijama. U praksi, DCP fajl predstavlja skup simboličkih informacija DCU fajlova jedinica koje su sadržane u paketu.

U vreme dizajniranja Delphiju su neophodni i BPL i DCU fajlovi jer prvi sadrži aktuelni kod komponenata kreiranih na formularu, a simboličke informacije su neophodne za tehnologiju Code Insight. Ukoliko paket dinamički povežete (koristeći ga kao paket samo za vreme izvršavanja), DCP fajl će takođe koristiti linker, a BPL fajl bi trebalo da se prosleđuje uz glavni izvršni fajl aplikacije. Ukoliko umesto toga paket statički povežete, linker se referiše na DCU fajlove i potrebno je da distribuirate samo konačni izvršni fajl.

Zbog toga, kao programer komponenata, trebalo bi, bar, da distribuirate BPL fajl, DCP fajl i DCU fajlove jedinica koje su sadržane u paketu i samo odgovarajuće DFM fajlove, i Help fajl. Kao opciju, naravno, možete učiniti dostupnim i izvorni kod fajlova jedinica paketa (PAS fajlove) i fajl samog paketa (DPK fajl).

Instaliranje komponenata ovog poglavlja

Kada smo izradili naš prvi paket, možemo početi da koristimo komponente koje smo u njega ugradili. Pre nego što to učinimo, potrebno je da pomenem da sam ja proširio paket MdPack da bih obuhvatio sve komponente koje ćemo izraditi u ovom poglavlju, uključujući različite verzije iste komponente. Ja Vam sugerišem da instalirate ovaj paket. Najbolji pristup je da kopirate paket u direktorijum putanje, tako da bude dostupan i Delphi okruženju i programima koje ćete uz pomoć paketa izraditi. Ja sam sakupio sve fajlove sa izvornim kodom komponenata i definicije

paketa u jedan direktorijum nazvanom MdPack. Ovo omogućava Delphi okruženju da se referiše samo na jedan direktorijum kada se traže paket i DCU fajlovi.

Zapamtite da ukoliko kompajlirate aplikaciju upotrebom paketa kao DLL-ova u vreme izvršavanja, potrebno je da instalirate ove nove biblioteke na klijent kompjutere. Ukoliko umesto toga kompajlirate programe statičkim povezivanjem paketa, DLL-ovi će biti neophodni samo za okruženje u kome se programira i neće biti potrebni korisnicima Vaših aplikacija.

NAPOMENA

Pored kreiranja i instaliranja pojedinačnih paketa, Delphi može da rukuje kolekcijama paketa. Package Collection Editor (PCE.EXE koji se nalazi u Delphi/Bin direktorijumu) omogućava Vam da više paketa smestite u jedan DPC (Delphi Package Collection) fajl. Ovaj fajl se zatim može instalirati u Delphi na isti način na koji instalirate samostalne pakete. Package Collection Editor Vam omogućava da odredite složenu instalaciju, sa nekoliko fajlova podrške, pa čak i da korisnicima omogućite da odaberu direktorijum u koji žele da instaliraju kompajlirane pakete, fajlove sa izvornim kodom i sve ostale fajlove podrške. ■

Upotreba combo polja Fonts

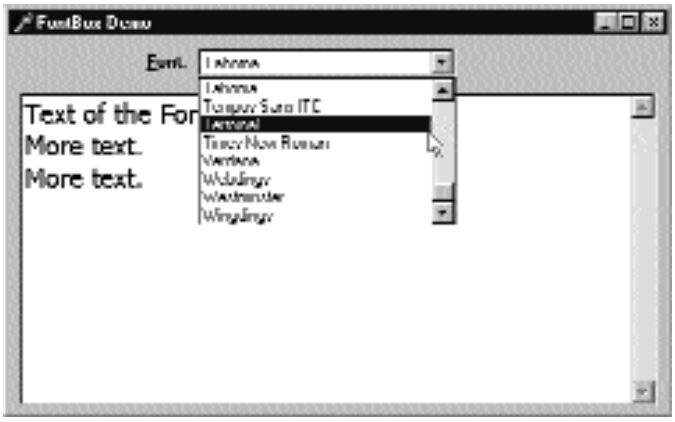
Sada možete da kreirate novi Delphi program da biste testirali combo polje Font. Pređite na Component Palette, odaberite novu komponentu i dodajte je na novi formular. Prikazaće se combo polje tradicionalnog izgleda. Ipak, ukoliko otvorite editor svojstva Items, videćete spisak svih fontova koji su instalirani na Vašem kompjuteru. Da bih izradio jednostavan primer, dodao sam Memo komponentu formularu u kojem se nalazi tekst. Program FontBoxDemo sadrži veoma malo koda. Kada korisnik odabere novi font iz combo polja, nova vrednost se koristi kao font Memo komponente:

```
procedure TForm1.MdFontCombo1Change (Sender: TObject);
begin
  // activate the new selection
  Memo1.Font.Name := MdFontCombo1.Text;
end;
```

Na početku se obavlja obrnuta akcija; naziv fonta Memo komponente prikazuje se u combo polju:

```
procedure TForm1.FormCreate (Sender: TObject);
begin
  // select the item corresponding to the current font
  MdFontCombo1.ItemIndex :=
    MdFontCombo1.Items.IndexOf (Memo1.Font.Name);
end;
```

Namena ovog jednostavnog programa (na slici 13.4 je izlaz ovog programa) je samo testiranje nove komponente koju smo izradili. Komponenta je još uvek veoma korisna — mogli smo da dodamo nekoliko linija koda formularu da bismo dobili isti efekat — ali to što smo pogledali nekoliko jednostavnih komponenata trebalo bi da nam da ideju šta je sve potrebno za izradu komponente.



SLIKA 13.4 *Izlaz primera FontBoxDemo*

Kreiranje složenih komponenata

Sledeća komponenta na koju ću obratiti pažnju je digitalni časovnik. Ovaj primer sadrži nekoliko interesantnih karakteristika. Prvo, ugnežđuje komponentu (Timer) u drugu komponentu; drugo, pokazuje pristup aktuelnih podataka (live-data).

Pošto digitalni časovnik daje nekakav tekstualni izlaz, ja sam u obzir uzeo nasleđivanje od klase `TLabel`. Ipak, to bi korisniku omogućilo da promeni tekst labele — to jest, tekst časovnika. Da bih izbegao ovaj problem, ja sam jednostavno upotrebio komponentu `TCustomLabel` roditeljske klase. Objekat `TCustomLabel` ima iste mogućnosti kao i objekat `TLabel`, ali i nekoliko published svojstava. Drugim rečima, potklasa `TCustomLabel` može odrediti koja svojstva bi trebalo da budu dostupna, a koja treba da ostanu sakrivena.

NAPOMENA

Većina Delphi komponenata, naročito Windows komponente, sadrže osnovnu klasu `TCustomXxx`, koja implementira svu funkcionalnost, ali prikazuje samo ograničeni skup svojstava. Nasleđivanje od ovih osnovnih klasa je standardni način za prikazivanje samo nekih svojstava komponente u prilagođenoj verziji. Zapravo, ne možete sakriti svojstva public ili published osnovne klase. ■

Pored ponovnog deklarisanja svojstava osnovne klase, `TMdClock` sadrži novo svojstvo `Active`. Ovo svojstvo određuje da li časovnik radi ili ne. Kao što ste mogli da pretpostavite, časovnik sadrži komponentu `TTimer`. Tajmer nije učinjen javnim preko svojstva jer ja nisam želeo da programeri mogu direktno da mu pristupe. Umesto toga, ja sam načinio dostupnim svojstvo `Enabled` komponente `Timer`, smeštajući ga unutar svojstva `Active` digitalnog časovnika. Evo cele deklaracije časovnika:

```
type
  TMdClock = class (TCustomLabel)
  private
    FTimer: TTimer;
    function GetActive: Boolean;
```

```
    procedure SetActive (Value: Boolean);
  protected
    procedure UpdateClock (Sender: TObject);
  public
    constructor Create (AOwner: TComponent); override;
  published
    property Align;
    property Alignment;
    property Color;
    property Font;
    property ParentColor;
    property ParentFont;
    property ParentShowHint;
    property. PopupMenu;
    property ShowHint;
    property Transparent;
    property Visible;
    property Active: Boolean
      read GetActive write SetActive;
  end;
```

Primetićete da su nam potrebni metodi za čitanje i pisanje vrednosti svojstva `Active`, jer vrednost svojstva nije lokalni podatak, već se referiše na član koji je ugnežđen u komponentu, dakle na Timer:

```
function TMdC1ock.GetActive: Boolean;
begin
  Result := FTimer.Enabled;
end;

procedure TMdClock.SetActive (Value: Boolean);
begin
  FTimer.Enabled := Value;
end;
```

Da bismo kreirali Timer, potrebno je da zaobiđemo konstruktor komponente časovnika. Metod `Create` poziva odgovarajući metod osnovne klase i kreira objekat Timer instalirajući obradu događaja `OnTimer`:

```
constructor TMdClock.Create (AOwner: TComponent);
begin
  inherited Create (AOwner);
  // create the internal timer object
  FTimer := TTimer.Create (Self);
  FTimer.OnTimer := UpdateClock;
  FTimer.Enabled := True;
end;
```

Kodom se ne određuje vrednost svojstva `Interval` tajmera jer je unapred određeni interval tajmera od 1000 milisekundi odgovarajući. Nije nam potreban destruktor jer je komponenta `TMDClock` vlasnik za objekat FTimer (što je naznačeno parametrom njegovog konstruktora `Create`), te će biti automatski uklonjen kada se ukloni komponenta časovnika.

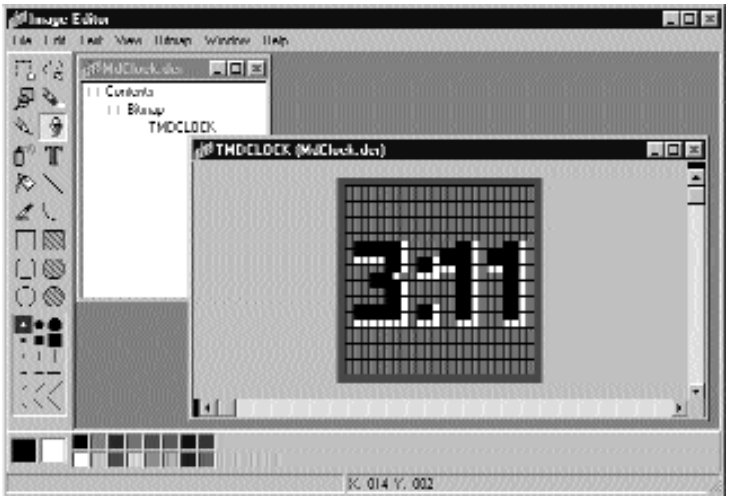
Ključni deo koda komponente je procedura `UpdateClock` koja sadrži samo jedan iskaz:

```
procedure TMdLabelClock.UpdateClock (Sender: TObject);
begin
  // set the current time as caption
  Caption := TimeToStr (Time):
end;
```

Ovaj metod koristi `Caption`, a to je svojstvo koje nije published, tako da ga korisnik komponente ne može izmeniti u Object Inspectoru. Rezultat ovog iskaza je prikazivanje trenutnog vremena. Ovo se stalno odvija jer je metod povezan sa događajem tajmera `OnTimer`.

Bitmape Component Palette

Pre instaliranja ove druge komponente možemo izvršiti još jedan korak: definisati bitmapu za Component Palette. Ukoliko to ne učinimo, Component Palette će upotrebiti bitmapu roditeljske klase, ili unapred određenu bitmapu objekta ukoliko roditeljska klasa nije instalirana komponenta (kao u slučaju komponente `TcustomLabel`). Definisanje nove bitmape za komponentu je lako kada jednom naučite pravila. Bitmapu možete kreirati upotrebom Image Editora (kao što je pokazano na slici 13.5), pokretanjem novog projekta i izborom DCR (Delphi Component Resource) tipa projekta.



SLIKA 13.5 *Definisanje bitmape za Component Palette u Delphijevom Image Editoru*

SAVET

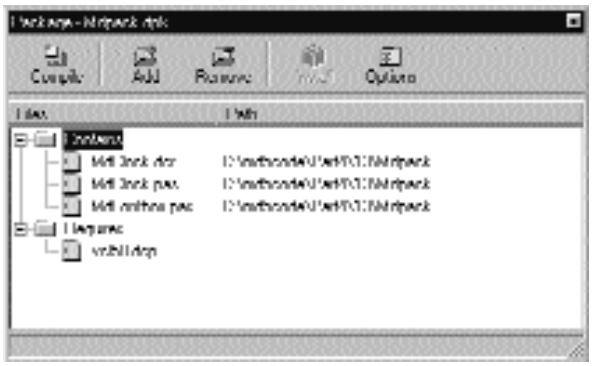
DCR fajlovi su samo standardni RES fajlovi sa drugačijom ekstenzijom. Ukoliko više volite, možete ih kreirati upotrebom resurs editora, uključujući Borland Resource Workshop, koji je svakako mnogo moćniji alat od Delphijevog Image Editora. Kada završite kreiranje resurs fajla, jednostavno preimenujte RES fajl tako da esktenzija bude DCR. ■

Sada novu bitmapu možemo da dodamo resursu, izborom veličine 24 x 24 piksela, i spremni smo da iscrtamo bitmapu. Drugo važno pravilo se odnosi na imenovanje. U ovom slučaju

pravilo imenovanja nije samo konvencija; to je obavezno da bi IDE mogao pronaći sliku za datu klasu komponente.

- Naziv resursa bitmape se mora poklopiti sa nazivom komponente, uključujući početno slovo T. U ovom slučaju naziv resursa bitmape treba da bude TMDCLOCK. Naziv resursa bitmape mora biti napisan velikim slovima. Ovo je obavezno.
- Ukoliko želite da Package Editor prepozna i uključi resurs fajl, naziv DCR fajla se mora podudariti sa nazivom kompajlirane jedinice koja definiše komponentu. U ovom slučaju naziv treba da bude MdClock.DCR. Ukoliko ručno uključite resurs fajl, upotrebom direktive `$R`, možete joj dodeliti naziv kakav želite, a možete, takođe, koristiti RES ili DCR fajlove sa više ikona.

Kada je bitmapa komponente spremna, komponentu možete da instalirate u Delphi upotrebom Package Editorove kontrole sa palete alata Install Package. Posle ove operacije, sekcija Contains editora bi trebalo da prikaže PAS fajl komponente i odgovarajući DCR fajl. Na slici 13.6 možete videti sve fajlove (uključujući DCR fajlove) konačne verzije paketa MdPack. Ukoliko DCR instalacija ne funkcioniše pravilno, možete ručno dodati iskaz `{$R unitname.dcr}` u izvorni kod paketa.



SLIKA 13.6 *Sekcija Contains Package Editora prikazuje obe jedinice koje su deo paketa i resurs fajlove komponente*

IZRADA SLOŽENIH KOMPONENATA UPOTREBOM OKVIRA

Umesto izrade složenih komponenata kodom i ručnog povezivanja događaja tajmera, sličan efekat možete postići upotrebom okvira. Okviri čine programiranje složenih komponenata sa korisničkim obradama događaja vizuelnom operacijom, te stoga i mnogo jednostavnijom. Ovaj okvir možete deliti dodajući ga u Respository ili kreiranjem šablona upotrebom komande Add to Palette iskačućeg menija okvira.

Alternativno, možda želite da okvir delite tako što ćete ga smestiti u paket i registrovati ga kao komponentu. Tehnički, ovo nije naročito teško. Potrebno je da dodate proceduru `Register` u jedinicu okvira, dodate jedinicu u paket i izradite ga. Component Palette će prikazati novu komponentu/okvir kao i bilo koju drugu komponentu. Ipak, kada ovu komponentu/okvir postavite na formular, nećete

videti njene potkomponente i nećete moći da sa njima komunicirate u vreme dizajniranja. Program koji se izvršava će se ponašati kako se očekuje, ali ograničena podrška u vreme dizajniranja čini ovaj pristup manje idelanim.

Aktivna kontrola

Windows interfejs evoluira ka novom standardu, uključujući komponente koje postaju istaknute kada se iznad njih nađe pokazivač miša. Delphi obezbeđuje sličnu podršku u mnogim svojim ugrađenim komponentama. Međutim, šta je potrebno da oponašate ovakvo ponašanje kada je u pitanju jednostavna kontrola koju Vi kreirate? Ovakvo ponašanje može izgledati kao složen zadatak, ali to, zapravo, nije.

Obično programiranje komponente može biti neverovatno jednostavno kada saznate koju virtuelnu funkciju treba zaobići, ili sa kojom porukom se treba povezati. Sledeća komponenta, klasa `TMdActiveButton`, demonstrira ovo obradom nekih internih Delphi poruka da bi svoj zadatak obavila na veoma jednostavan način.

NAPOMENA

Kako sam ja odredio koje poruke su one koje treba upotrebiti, kada su one gotovo potpuno nedokumentovane? Proučavanjem VCL izvornog koda. To je, uopšte, veoma dobar način da postanete ekspert za izradu komponenata. Takođe bi trebalo da proučite neke napredne knjige o Delphiju. ■

Komponenta ActiveButton obrađuje interne Delphi poruke `cm_MouseEnter` i `cm_MouseExit` koje se dobijaju kada pokazivač miša uđe u oblast, ili izađe iz oblasti koja odgovara komponenti:

```
type
  TMdActiveButton = class (TButton)
  protected
    procedure MouseEnter (var Msg: TMessage);
      message cm_mouseEnter;
    procedure MouseLeave (var Msg: TMessage);
      message cm_mouseLeave;
  end;
```

Kod koji pišete za ova dva metoda može učiniti bilo šta što poželite. Za ovaj primer sam odlučio da menjam podebljana slova same komponente. Efekat pomeranja pokazivača miša iznad jedne od ovih komponenata možete videti na slici 13.7.

```
procedure TMdActiveButton.MouseEnter (var Msg: TMessage);
begin
  Font.Style := Font.Style + [fsBold];
end;

procedure TMdActiveButton.MouseLeave (var Msg: TMessage);
begin
  Font.Style := Font.Style - [fsBold];
end;
```

Po želji možete dodati i druge efekte, uključujući i povećanje samog fonta, čineći kontrolu unapred definisanom ili povećavajući njenu veličinu. Najbolji efekti obično se postižu dodavanjem boje, ali

morate nasleđivati iz klase `TBitBtn` da biste imali podršku za boje (kontrole `TButton` imaju fiksiranu boju).

SLIKA 13.7 *Primer upotrebe komponente ActiveButton*

Složene grafičke komponente

Grafička komponenta koju želim da izradim jeste komponenta strelice. Ovakvu kontrolu možete upotrebiti da biste naznačili, na primer, tok informacija ili akciju. Ovakva komponenta je prilično složena te ću Vam pokazati različite potrebne korake umesto da Vam odmah pokažem kod komponente. Komponenta koju sam dodao paketu MdPack samo je finalna verzija ovog procesa, koji će prikazati brojne važne koncepte:

- Definicija novih pobrojanih svojstava koja se zasnivaju na korisničkim pobrojanim tipovima podataka.
- Upotreba svojstava klasa izvedenih iz klase `TPersistent`, kao što su klase `TPen` i `TBrush` i stvari koje se tiču njihovog kreiranja i uklanjanja, i obrada njihovih `OnChange` svojstava interno u okviru naše komponente.
- Implementacija metoda `Paint` komponente, koji obezbeđuje korisnički interfejs komponente i koji bi trebalo da bude dovoljno generički da bi prihvatio sve moguće vrednosti različitih svojstava, uključujući `Width` i `Height`. Metod `Paint` igra značajnu ulogu za ovu grafičku komponentu.
- Definicija korisničke obrade događaja za komponentu, koja reaguje na korisnički unos (u ovom slučaju kada se dva puta klikne strelica). Ovo će zahtevati direktnu obradu Windows poruka i upotrebu Windows API-ja za grafičke regione.
- Registracija svojstava u kategorijama Object Inspectora i definicija naše kategorije.

Definisanje pobrojanog svojstva

Posle generisanja nove komponente upotrebom Component Wizarda i izborom klase `TGraphicControl` za roditeljsku, možemo početi da prilagođavamo komponente. Strelica može da pokazuje bilo koji od četiri pravca: gore, dole, levo i desno. Pobrojani tip izražava ove mogućnosti:

```
type
  TMdArrowDir = (adUp, adRight, adDown, adLeft);
```

Ovaj pobrojani tip definiše privatnog člana podataka komponente, parametar procedure koji se koristi da bi se promenio član i tip odgovarajućeg svojstva. Još dva jednostavna svojstva su `ArrowHight` i `Filled`; prvo određuje veličinu vrha strelice, a drugo da li je strelica ispunjena bojom:

```
TMdArrow = class (TGraphicControl)
  private
    fDirection: TMdArrowDir;
    fArrowHeight: Integer;
    fFilled: Boolean;
    procedure SetDirection (Value: TMd4ArrowDir);
    procedure SetArrowHeight (Value: Integer);
    procedure SetFilled (Value: Boolean);
  published
    property Width default 50;
    property Height default 20;
    property Direction: TMd4ArrowDir
      read fDirection write SetDirection default adRight;
    property ArrowHeight: Integer
      read fArrowHeight write SetArrowHeight default 10;
    property Filled: Boolean
      read fFilled write SetFilled default False;
```

NAPOMENA

Grafička kontrola nema unapred određenu veličinu i kada je postavite na formular, njena veličina će biti samo jedan piksel. Zbog toga je veoma važno odrediti unapred određenu vrednost za svojstva `Width` i `Height` i da za polja klase odredite unapred određene vrednosti svojstava u konstruktoru klase. ■

Tri korisnička svojstva se direktno čitaju iz odgovarajućeg polja i zapisuju se pomoću tri Set metoda, koji svi imaju istu standardnu proceduru:

```
procedure TMdArrow.SetDirection (Value: TMdArrowDir);
begin
  if fDirection <> Value then
  begin
    fDirection := Value;
    ComputePoints;
    Invalidate;
  end;
end;
```

Primetićete da od sistema tražimo da ponovo iscrta kontrolu (pozivom `Invalidate`) samo ukoliko svojstvo zaista menja svoju vrednost i posle poziva metoda `ComputePoints`, koji izračunava

trougao koji odvaja vrh strlice. U suprotnom, kod se preskače i metod se odmah završava. Ovakva struktura koda je veoma uobičajena i koristićemo je za većinu Set procedura svojstava.

Takođe, ne smemo zaboraviti da odredimo unapred određene vrednosti svojstava konstruktora komponente:

```
constructor TMdArrow.Create (AOwner: TComponent);
begin
  // call the parent constructor
  inherited Create (AQwner);
  // set the default values
  fDirection := adRight;
  Width := 50;
  Height  := 20;
  fArrowHeight := 10;
  fFilled := False;
```

Zapravo, kao što sam ranije istakao, unapred podešena vrednost koja je određena deklaracijom svojstva, koristi se samo da bi se odredilo da li da se vrednost svojstva sačuva na disku. Konstruktor `Create` je definisan u javnoj (public) sekciji definicije tipa nove komponente i naznačen je ključnom reči `override`. Veoma je važno zapamtiti ovu ključnu reč; u suprotnom, kada Delphi kreira novu komponentu ove klase, Delphi će pozvati konstruktor osnovne klase, a ne konstruktor koji ste napisali za izvedenu klasu.

KONVENCIJE IMENOVANJA SVOJSTAVA

U definiciji komponente Arrow primetićete upotrebu nekoliko konvencija imenovanja svojstava, metoda pristupanja i polja. Evo rezimea:

- Svojstvo treba da ima naziv koji je intuitivan i lako čitljiv.
- Kada se koristi privatno (**private**) polje podataka za čuvanje vrednosti svojstva, polju treba dati naziv koji počinje slovom f (field — polje) za kojim sledi naziv odgovarajućeg svojstva.
- Kada se koristi funkcija za promenu vrednosti svojstva, naziv funkcije treba počinjati rečju Set za kojom sledi naziv same funkcije.
- Odgovarajuća funkcija koja se koristi za čitanje svojstva treba da na početku ima reč Get za kojom sledi naziv svojstva.

Ovo su samo neke smernice koje bi program trebalo da učine čitljivijim. Kompajler ih ne zahteva. Ove konvencije su opisane u knjizi Delphi Component Writers Guide (Priručnik za programere Delphi komponenata) i sledi ih Delphijev mehanizam kompletiranja klasa.

Pisanje metoda Paint

Iscrtavanje strelice u različitim pravcima i sa različitim stilovima zahteva priličnu količinu koda. Da biste izvršili korisničko iscrtavanje, potrebno je da zaobiđete metod `Paint` i upotrebite zaštićeno svojstvo `Canvas`.

Umesto izračunavanja pozicije glave strelice (u pointima) u kodu za iscrtavanje koji će se često izvršavati, napisao sam zasebnu funkciju za izračunavanje oblasti glave strelice i za čuvanje u nizu tačaka definisanih među privatnim poljima na sledeći način:

```
fArrowPoints: array [0..3] of TPoint;
```

Ove tačke su određene privatnim metodom `ComputePoints` koji se poziva svaki put kada se neka svojstva komponente promene. Evo dela koda ovog metoda:

```
procedure TMdArrow.ComputePoints;
var
  XCenter, YCenter: Integer;
begin
  // caupute the paints at the arrowhead
  YCenter := (Height - 1) div 2;
  XCenter := (Width - 1) div 2;
  case FDirection of
    adUp: begin
      fArrowPoints [0] := Point (0, FArrowHeight);
      fArrowPoints [1] := Point (XCenter, 0);
      fArrowPoints [2] := Point (Width - 1, FArrowHeight);
    end;
  // and so on for the other directions
```

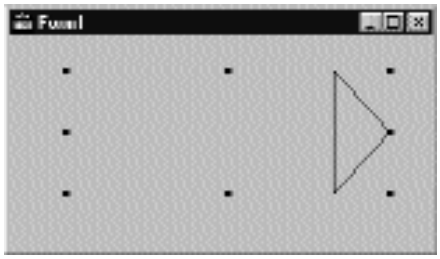
Kod izračunava centar oblasti komponente (jednostavnim deljenjem svojstava `Height` i `Width` sa dva), a zatim pomoću centra određuje poziciju glave strelice. Pored promene smera i drugih svojstava, potrebno je da osvežimo poziciju glave strelice kada se promeni veličina komponente. Ono što možemo učiniti jeste da zaobiđemo metod `SetBounds` ove komponente, koji VCL poziva svaki put kada se promene svojstva komponente `Left`, `Top`, `Width` i `Height`:

```
procedure TMdArrow.SetBounds(ALeft, ATop, AWidth, AHeight: Integer);
begin
  inherited SetBounds (ALeft, ATop, AWidth, AHeight);
  ComputePoints;
end;
```

Kada komponenta sazna poziciju glave strelice, njen kod iscrtvanja postaje jednostavniji. Evo dela metoda `Paint`:

```
procedure TMdArrow. Paint;
var
  XCenter, YCenter: integer;
begin
  // compute the center
  YCenter := (Height - 1) div 2;
  XCenter := (Width - 1) div 2;

  // draw the arrow line
  case FDirection of
    adUp: begin
      Canvas.MoveTo (XCenter, Height - 1);
      Canvas.LineTo (XCenter, FArrowHeight);
    end;
    // and so on for the other directions
  end;
```

```
  // draw the arrow point, eventually filling it
  if FFilled then
    Canvas.Polygon (fArrowPoints);
  else
    Canvas.PolyLine (fArrowPoints);
end;
```



SLIKA 13.8 *Izlaz komponente Arrow*

Dodavanje svojstava TPersistent

Da bi izlaz komponente bio fleksibilniji, ja sam dodao komponenti dva nova svojstva definisana tipom klase (preciznije, tipom podataka `TPersistent`, koji definiše objekte koji se mogu automatski ulančiti u Delphi). Ovakvim svojstvima se nešto teže rukuje jer komponenta sada mora da kreira i uklanja ove interne objekte (kao što smo to činili sa internom komponenteom Timer komponente časovnika). Ovoga puta mi takođe izvozimo interne objekte upotrebom nekih svojstava tako da korisnici mogu direktno da ih menjaju iz Object Inspectora. Da bismo ažurirali komponentu kada se ovi podobjekti promene, takođe je potrebno da obradimo njihovo interno svojstvo `OnChange`. Evo definicija dva nova svojstva `TPersistent` i druge izmene definicije klase komponente:

```
type
    TMdArrow = class (TGraphicControl)
  private
    FPen: TPen;
    FBrush: TBrush;
    ...
    procedure SetPen (Value: TPen);
    procedure SetBrush (Value: TBrush);
    procedure RepaintRequest (Sender: TObject);
  published
    property Pen: TPen
    read FPen write SetPen;
  property Brush: TBrush
    read FBrush write SetBrush;
end;
```

Prva stvar koju treba uraditi je kreiranje objekata u konstruktoru i određivanje njihovih obrada događaja `OnChange`:

```
constructor TMdArrow.Create (AOwner: TComponent);
begin
  ...
  // create the pen and the brush
  FPen := TPen.Create;
  FBrush := TBrush.Create;
  // set a handler for the OnChange event
  FPen.OnChange := RepaintRequest;
  FBrush.OnChange := RepaintRequest;
end;
```

Ovi događaji `OnChange` se iniciraju kada se jedno od svojstava ovih podsvojstava promeni; sve što treba da učinimo je da zatražimo od sistema da ponovo iscrta našu komponentu:

```
procedure TMdArrow.RepaintRequest (Sender: TObject);
begin
  Invalidate;
end;
```

Takođe, morate dodati destruktor komponente da biste uklonili dva grafička objekta iz memorije (i da biste oslobodili sistemske resurse):

```
destructor TMdArrow.Destroy;
begin
  FPen.Free;
  FBrush.Free;
  Inherited Destroy;
end;
```

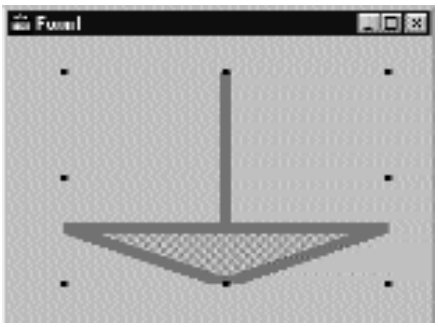
Svojstva koja su u vezi sa ovim dvema komponentama zahtevaju specijalnu obradu: umesto kopiranja pokazivača objekata potrebno je da kopiramo interne podatke objekta koji se prosleđuju kao parametar. Standardna operacija `:=` kopira pokazivače, te je u ovom slučaju potrebno da koristimo metod `Assign`. Evo jedne od dveju Set procedura:

```
procedure TMdArrow.SetPen (Value: TPen);
begin
  FPen.Assign(Value);
  Invalidate;
end;
```

Mnoge `TPersistent` klase sadrže metod `Assign` koji treba da koristimo kada je potrebno da ažuriramo podatke ovih objekata. Sada, da bismo zaista upotrebili olovku i četkicu za crtanje, potrebno je da izmenite metod `Paint` podešavajući svojstva `Pen` i `Brush` komponente `Canvas` na vrednost internih objekata pre iscrtavanja bilo koje linije:

```
procedure TMdArrow.Paint;
begin
  // use the current pen and brush
  Canvas.Pen := FPen;
  Canvas.Brush := FBrush;
```

Primer novog izlaza komponente je prikazan na slici 13.9.



SLIKA 13.9 *Izlaz komponente Arrow debljom olovkom i specijalnom četkicom*

Definisanje novog korisničkog događaja

Da bismo dovršili razvoj komponente Arrow, dodajmo još i događaj. Veći deo vremena nove komponente koriste događaje svojih roditeljskih klasa. Na primer, u ovoj komponenti sam učinio dostupnim standardne događaje putem jednostavnog ponovnog njihovog deklarisanja u published sekciji klase:

```
type
  TMdArrow = class (TGraphicControl)
  published
    property OnClick;
    property OnDragDrop;
    property OnDragOver;
    property OnEndDrag;
    property OnMouseDown;
    property OnMouseMove;
    property OnMouseUp;
```

Zahvaljujući ovoj deklaraciji, prethodni događaji (originalno deklarisani u roditeljskoj klasi) biće dostupni iz Object Inspectora kada se komponenta instalira.

Ponekad komponenta zahteva korisnički događaj. Da biste definisali potpuno novi događaj, prvo je potrebno da klasi dodate polje tipa događaja. Ovaj tip je, zapravo, tip pokazivača događaja (pogledajte Poglavlje 3 za više detalja). Ovo je definicija koju sam dodao pivatnom odeljku klase `TMdArrow`:

```
fArrowDblClick: TNotifyEvent
```

U ovom slučaju sam koristio tip `TNotifyEvent` koji ima samo parametar `Sender`, a Delphi ga koristi za mnoge događaje, uključujući događaje `OnClick` i `OnDblClick`. Upotrebom ovog polja definisao sam veoma jednostavno javno svojstvo sa direktnim pristupom polju:

```
property OnArrowDblClick: TNotifyEvent
  read fArrowDblClick write fArrowDblClick;
```

Ponovo ćete primetiti standardne konvencije imenovanja kada imena događaja počinju sa *On*. Pokazivač metoda `fArrowDblClick` se aktivira (izvršavajući odgovarajuću funkciju) unutar speci-

fičnog dinamičkog metoda ArrowDblClick. Ovo se dešava samo ukoliko je obrada događaja naznačena u programu koji koristi komponentu:

```
procedure TMdArrow.ArrowDblClick;
begin
  if Assigned (FArrowDblClick) then
    FArrowDblClick (Self);
end;
```

Ovaj metod je definisan u zaštićenom odeljku definicije tipa da bi budućim potklasama omogućio poziv i izmene. U osnovi, metod ArrowDblClick se poziva iz obrade wm_LButtonDblClick Windows poruke, ali samo ukoliko se dvostruki klik dogodi unutar glave strelice. Da bi se proverio ovaj uslov, možemo upotrebiti neke od Windows API funkcija regiona.

NAPOMENA

Region je oblast ekrana ograničena bilo kojim oblikom. Na primer, možemo nacrtati poligonalni region upotrebom tri duži trougla glave strelice. Jedini problem je u tome što — da bismo pravilno ispunili površinu — potrebno je da definišemo niz tipa TPoints u smeru suprotnom od smera kazaljke na satu (pogledajte opis CreatePolygonalRgn u Windows API Helpu za detalje o ovom čudnom pristupu). To je ono što sam ja učinio u metodu ComputePoints. ■

Kada smo definisali region, upotrebom poziva PtInRegion API funkcije možemo testirati da li se tačka gde ste dva puta kliknuli nalazi unutar regiona. Ceo izvorni kod **procedure** možete videti u narednom listingu:

```
procedure TMdArrow.WMLButtonDblClk (
  var Msg: TWMLButtonDblClk); // message wm_LButtonDblClk;
var
  HRegion: HRgn;
begin
  // perform default handling
  inherited;

  // compute the arrowhead region
  HRegion := CreatePolygonRgn (
    fArrowPoints, 3, WINDING);
  try
    // check ehether the click took place in the region
    if PtInRegion (HRegion, Msg.XPos, Msg.YPos) then
      ArrowDblClk;
  finally
    DeleteObject (HRegion);
  end;
end;
```

Registrovanje kategorija svojstava

Ovoj komponenti smo dodali neka naša svojstva i novi događaj. Ukoliko u Object Inspectoru svojstva uredite po kategoriji, svi novi elementi će se naći u generičkoj kategoriji Miscellaneous. Naravno, ovo je daleko od idealnog, ali mi možemo nova svojstva lako registrovati u jednoj od postojećih kategorija (navedene su u Delphijevom Help fajlu).

Mi možemo registrovati svojstvo (ili događaj) u kategoriji pozivom jedne od četiri preopterećene verzije funkcije `RegisterPropertyInCategory` (koje su definisane u jedinici DsgnIntf) i navođenjem naziva svojstva, njegovog tipa ili naziva svojstva i komponente kojoj pripada. Na primer, možemo dodati sledeće linije proceduri `Register` jedinice da bismo registrovali događaj `OnArrowDblClick` u kategoriji Input i svojstvo `Filled` u kategoriji Visual:

```
uses
  DsgnIntf;

procedure Register;
begin
  RegisterComponents ('Md', [TMdArrow]);
  RegisterPropertyInCategory (
    TInputCategory, TMdArrow, 'OnArrowDblClick');
  RegisterPropertyInCategory (
    TVisualCategory, TMdArrow, ' ');
end;
```

Takođe, možemo učiniti još jednu stvar i kreirati potpuno novu kategoriju za specifična svojstva naše komponente, što će korisnicima umnogome olakšati da pronađu specifična svojstva. Da bismo ovo postigli, jednostavno ćemo novu klasu izvesti iz generičke klase `TPropertyCategory` i zaobići funkciju klase `Name`. Name će se koristiti da bi se naznačla kategorija u Object Inspectoru, dok se druga funkcija koju zaobilazimo, `Description`, očigledno ne koristi.

UPOZORENJE

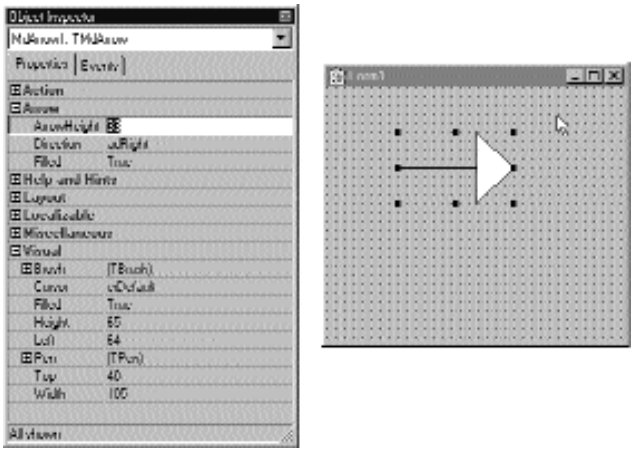
Može se raspravljati da li je upotreba kategorija za specifična svojstva dobra ideja. S jedne strane, korisnik komponente može lako uočiti specifična svojstva. Istovremeno, neka od novih svojstava možda ne spadaju ni u jednu od postojećih kategorija. S druge strane, kategorije se mogu previše upotrebljavati. Kada bi svaka komponenta uvodila nove kategorije, korisnici bi mogli da se zbune. Takođe, postoji rizik da imate onoliko kategorija koliko imate svojstava. Pošto je ovo potpuno novi alat u Delphiju 5, moramo sačekati i videti kako će programeri komponenata prihvatiti ovaj alat i koliko će se alat svideti korisnicima. ■

Evo koda kategorije svojstva koju sam definisao za komponentu Arrow:

```
interface

type
  TArrowCategory = class (TPropertyCategory)
    class function Name: string; override;
    class function Description: string; override;
end;

implementation

class function TArrowCategory.Name: string;
begin
  Result := 'Arrow';
end;

class function TArrowCategory.Description: string;
begin
  Result := 'Properties of the Mastering Delphi Arrow component';
```

```
end;

procedure Register;
begin
  ...
  RegisterPropertyInCategory (
    TArrowCategory, TMdArrow, 'Direction');
  RegisterPropertyInCategory (
    TArrowCategory, TMdArrow, 'ArrowHeight');
  RegisterPropertylnCategory (
    TArrowCategory, ThidArrow, 'Filled');
end;
```

Primetićete da je svojstvo `Filled` već bilo registrovano u drugoj postojećoj kategoriji. To ne predstavlja problem jer se isto svojstvo može prikazati više puta u Object Inspectoru pod različitim kategorijama, kao što možete videti na slici 13.10.



SLIKA 13.10 *Komponenta Arrow deifniše kategoriju svojstva Arrow, kao što možete videti u Object Inspectoru*

Da bih testirao komponentu Arrow ja sam napisao veoma jednostavan program, program ArrowDemo, koji Vam omogućava da izmenite većinu svojstava komponente u vreme izvršavanja. Ovakav tip testa, pošto ste napisali komponentu ili za vreme pisanja komponente, veoma je važan.

Prilagođavanje Windows kontrola

Jedan od najuobičajenijih načina prilagođavanja postojećih komponenata je da dodate unapred određeno ponašanje obradama događaja komponenata. Svaki put kada je potrebno da dodate obradu događaja komponentama različitih oblika, treba da razmislite o dodavanju koda događaja direktno u potklasu komponente. Očigledan primer su polja za izmene koja prihvataju samo numeričke vrednosti. Umesto pridruživanja obrade događaja `OnChange` svakom polju za izmene, možemo definisati jednostavnu novu komponentu. Ova komponenta ipak neće obrađivati događaj;

događaji su samo za korisnike komponenata. Umesto toga, komponenta može direktno da obradi Windows poruku ili zaobiđe metod, kao što smo opisali u prethodna dva odeljka.

Zaobilaženje obrada poruka: numeričko polje za izmene

Da biste prilagodili polje za izmene tako da ograničite mogući unos, sve što je potrebno da učinite jeste obrada Windows poruke `wm_Char` koja se javlja kada korisnik pritisne jedan od nekoliko specifičnih tastera (nazivaju se numerički karakteri).

Jedan način na koji možete odgovoriti na poruku za dati prozor (bilo da je u pitanju formular ili komponenta) jeste da kreirate novi metod koji reaguje na poruke (message-response method), koji ćete deklarisati ključnom reči `message`. Delphijev sistem obrade poruka će se postarati da Vaš metod za poruke dobije šansu da reaguje na datu poruku pre formulara ili unapred određene obrade poruke. Kao što ćemo videti u narednom odeljku, umesto kreiranja novog metoda (kao što ćemo to ovde učiniti) možete zaobići virtuelni metod koji reaguje na datu poruku. Pogledajte kompletan kod klase `TMdNumEdit`:

```
type
  TMdNumEdit = class (TCustomEdit)
private
  fInputError: TNotifyEvent;
protected
  function GetValue: Integer;
  procedure SetValue (Value: Integer);
public
  procedure WmChar (var Meg: TWmChar): message wm_Char;
  constructor Create (Owner: TComponent); override;
published
  property OnInputError: TNotityEvent
    read fInputError write fInputError;
  property Value: Integer
    read GetValue write SetValue default 0;
  property AutoSelect;
  property AutoSize;
  property BorderStyle;
  // and so on ...
```

Ova komponenta nasleđuje iz klase `TCustomEdit` umesto iz klase `TEdit` tako da može da sakrije svojstvo `Text` i da umesto toga prikaže svojstvo Value. Primetićete da ja ne kreiram novo polje za čuvanje ove vrednosti jer možemo da koristimo postojeće (ali sada javno) svojstvo `Text`. Da bismo ovo učinili, mi ćemo jednostavno konvertovati numeričku vrednost u string i string u numeričku vrednost. Klasa `TCustomEdit` (ili zapravo Windows kontrola koju obuhvata) automatski prikazuje informaciju iz svojstva `Text` na površini komponente:

```
function TMdNumEdit.GetValue: Integer;
begin
  // set to 0 in case of error
  Result := StrToIntDef (Text, 0);
end;

procedure TMdNumEdit.SetValue (Value: Integer);
begin
  Text := IntToStr (Value);
end;
```

Najvažniji metod je reagovanje na poruku wm_Char. U telu metoda komponenta izdvaja sve nenumeričke karaktere i poziva specifični događaj u slučaju greške:

```
procedure TMdNumEdit.WmChar (var Msg: TWmChar);
begin
  if not (Char (Msg.CharCode) in ['0'.. '9'])
    and not (Msg.CharCode = 8) then
  begin
    Msg.CharCode := 0;
    if Assigned (flnputError) then
      fInputError (Self);
  end;
end;
```

Ovaj metod proverava svaki karakter kako ga korisnik unosi, testirajući numerale i taster Backspace (čija je ASCII vrednost 8). Korisnik bi trebalo da ima mogućnost upotrebe tastera Backspace kao i sistemskih tastera (kursor tastera i tastera Delphi), te je potrebno da proverimo i tu vrednost. Nije potrebno da proveravamo sistemske vrednosti jer se izdvajaju drugom Windows porukom, porukom wm_SysChar.

To je sve. Ukoliko sada postavite komponentu na formular, možete uneti nešto u polje za izmene i videti kako se ponaša. Možda želite da dodate metod za događaj OnInputError da biste korisnika mogli da obavestite kada pritisne pogrešan taster.

Zaobilaženje dinamičkih metoda: kontrola Sound

Naša sledeća komponenta, TMdSoundButton, daje zvuk kada kliknete kontrolu i drugi zvuk kada otpustite taster miša. Korisnik bira zvuke izmenom dva String svojstva kojima se imenuju odgovarujći WAV fajlovi za zvuke. Ponovimo još jednom, potrebno je da presretnemo i izmenimo neke sistemske poruke (wm_LButtonDown i wm_LButtonUp), ali umesto da poruke obradimo tako što ćemo napisati novi metod za reagovanje na poruke, mi ćemo zaobići odgovarajuće obrade drugog nivoa (second-level handlers).

NAPOMENA

Kada većina VCL komponenata obrađuje Windows poruku, one pozivaju obradu događaja drugog nivoa (obično dinamički metod) umesto direktnog izvršavanja koda u metodu koji reaguje na poruke. Ovakav pristup Vam olakšava prilagođavanje komponente u izvedenoj klasi. Obrada drugog nivoa će obaviti sav svoj posao i zatim će pozvati obradu događaja koju je dodelio korisnik komponente. ■

Ovo je kod klase TMdSoundButton, sa dvema zaštićenim (protected) obradama drugog nivoa i dva string svojstva koja identifikuju zvučne fajlove. Primetićete da u deklaracijama svojstava pišemo i čitamo odgovarajuća privatna polja bez poziva metoda Get ili Set jer ne moramo da učinimo ništa specijalno kada korisnik izmeni ova svojstva.

```
type
  TMdSoundButton = class(TButton)
  private
    FSoundUp, PSoundDown: string;
  protected
    procedure MouseoDwn(Button: TMouseButton;
      Shift:  TShiftState; X, Y: Integer); override;
```

```
    procedure Mouseup(Button: TMouseButton;
      Shift:  TShiftState; X, Y: Integer); override;
    published
      property SoundUp: string
        read FSoundUp write FSoundUp;
      property SoundDown: string
        read FSoundDown write FSoundDown;
  end;
```

Postoji nekoliko razloga zašto je zaobilaženje postojećih obrada drugog nivoa, uopšte uzev, bolji pristup od direktne obrade Windows poruka. Prvo, ova tehnika je mnogo bolja iz objektno orijentisane perspektive. Umesto dupliranja koda za reagovanje na poruke osnovne klase i njegovog prilagođavanja, Vi zaobilazite poziv virtuelnog metoda koji su VCL programeri planirali da zaobiđete. Drugo, ukoliko neko želi da izvede neku drugu klasu iz Vaše klase komponente, želite da mu olakšate prilagođavanje koliko je god to moguće, a zaobilaženje obrada drugog nivoa će mnogo manje dovesti do čudnih grešaka (ako ništa drugo zato što pišete manje koda). Konačno, ovo će Vaše klase komponenata učiniti više konzistentnim sa VCL-om — i zato će ih neko drugi lakše shvatiti. Evo koda dveju obrada drugog nivoa:

```
uses
  MMSystem;

procedure TMdSoundButton.MouseDown(Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  inherited MouseDown (Button, Shift, X, Y);
  PlaySound (Pchar (FSoundDown), 0, snd_Async);
end;

procedure TMdSoundButton.MouseUp(Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  inherited MouseUp (Button, Shift, X, Y);
  PlaySound (PChar (FSoundUp), 0, snd_Async);
end;
```

U oba slučaja, primetićete da pozivamo nasleđenu verziju metoda pre nego što učinimo bilo šta drugo. Za većinu obrada drugog nivoa ovo je dobra praksa, jer obezbeđuje izvršavanje standardnog ponašanja pre izvršavanja bilo kakvog korisničkog ponašanja.

Zatim, primetićete da pozivamo Win32 API funkciju `PlaySound` da bismo proizveli zvuk. Ovu funkciju (koja je definisana u jedinici MmSystem) možete koristiti za puštanje WAV fajlova ili sistemskih zvukova, kao što to demonstrira primer SoundB. Evo tekstualnog opisa formulara ovog programa (iz DFM fajla):

```
object MdSoundButton1: TMdSoundButton
  Caption = 'Press'
  SoundUp = 'ResroreUp'
  SoundOown = 'RestoreDown'
end
```

NAPOMENA

Pravilno selektovanje vrednosti za ova zvučna svojstva je sve samo ne jednostavno. Kasnije u ovom poglavlju ću Vam pokazati kako da dodate editor svojstva komponenti da biste operaciju učinili jednostavnijom. ■

Nevizuelna komponenta Dialog

Sledeća komponenta kojom ćemo se pozabaviti potpuno se razlikuje od komponenata koje smo do sada videli. Posle izrade kontrola koje koriste prozore, i jednostavnih grafičkih komponenata, sada ću izraditi nevizuelnu komponentu.

Osnovna ideja je da su formulari komponente. Kada je potrebno da izradite formular koji može biti naročito koristan za veliki broj projekata, možete ga dodati u Object Repository ili možete od njega načiniti komponentu. Drugi pristup je mnogo složeniji od prvog, ali olakšava upotrebu novog formulara i omogućava Vam da novi formular distribuirate bez njegovog koda. Kao primer ću izraditi komponentu koja se zasniva na uobičajenom okviru za dijalog, i pokušaću da, što je više moguće, oponašam ponašanje standardne Delphi komponente okvira za dijalog.

Prvi korak u pretvaranju okvira za dijalog u komponentu jeste pisanje koda samog okvira za dijalog upotrebom standardnog Delphi pristupa. Jednostavno definišite novi formular i na njemu radite kao i obično. Kada je komponenta bazirana na formularu, možete gotovo vizuelno da dizajnirate komponentu. Naravno, kada se izradi okvir za dijalog, morate oko njega definisati komponentu na nevizuelni način.

Standardni okvir za dijalog koji želim da izradim bazira se na listi, jer je uobičajeno da korisniku omogućite da izabere vrednost iz liste stringova. Ja sam prilagodio ponašanje ove komponente u okviru za dijalog, a zatim sam ga upotrebio za izradu komponente. Jednostavan formular `ListBoxForm` koji sam izradio sadrži listu i uobičajene kontrole OK i Cancel, kao što je pokazano sledećim tekstualnim opisom:

```
object MdListBoxForm: TMdListBoxForm
  BorderStyle = bsDialog
  Caption = 'LtstBoxForn'
  object ListBox1: TListBox
    OnDblClick = ListBox1DblClick
  end
  object BitBtn1: TBitBtn
    Kind = bkOK
  end
  object BitBtn2: TBitBtn
    Kind = bkCancel
  end
end
```

Jedini metod ovog okvira za dijalog odnosi se na događaj kada korisnik dva puta klikne listu, čime se zatvara okvir za dijalog, što je isto kao da je korisnik kliknuo kontrolu OK:

```
procedure TMdListBoxForm.ListBox1DblClick(Sender: TObject);
begin
  ModalResult := mrOk;
end;
```

Kada formular proradi, možemo početi da menjamo izvorni kod formulara, dodajemo definicije komponente i uklanjamo deklaracije globalne promenljive za formular.

NAPOMENA

Za komponente koje se zasnivaju na formularu, možete koristiti dva fajla sa Pascal izvornim kodom: jedan je fajl formulara, a drugi za komponentu koju enkapsulirate. Takođe je moguće smestiti i formular i komponentu u istu jedinicu, kao što sam ja učinio u ovom primeru. Teoretski bi bilo mnogo lepše deklarisati formular klase u implementacionom delu jedinice, sakrivajući je od korisnika komponente. U praksi ovo nije dobra ideja. Da biste formularom vizuelno manipulisali iz Form Designera, deklaracija formular klase se mora pojaviti u interfejs odeljku jedinice. Razlog za ovakvo ponašanje Delphi IDE-a je što veze minimizuju veličinu koda koji menadžer modula mora da pretražuje da bi pronašao deklaraciju formulara — operacija koja često mora da se obavlja da bi se održala sinhronizacija vizeulenog formulara sa definicijom formular klase. ■

Najvažnija od ovih operacija je definicija komponente `TMdListBoxDialog`, dakle nevizuelne komponente. Ova komponenta je nevizuelna jer je klasa `TComponent` njen direktni roditelj. Komponenta sadrži tri published svojstva i jedno javno (public). Tri published svojstva su sledeća:

- Lines je objekat `TString`, kojem se pristupa preko dva metoda, `GetLines` i `SetLines`. Drugi metod koristi proceduru `Assign` za kopiranje novih vrednosti u privatno polje koje odgovara ovom svojstvu. Ovaj interni objekat se inicijalizuje konstruktorom `Create` i uklanja metodom `Destroy`.
- `Selected` je celobrojna vrednost koja direktno pristupa odgovarajućem privatnom polju. Čuva selektovane elemente liste stringova.
- `Title` je string koji se koristi za promenu naslova okvira za dijalog.

Javno svojstvo je `SelItem`, svojstvo samo za čitanje, koje automatski daje selektovani element liste stringova. Primetićete da ovo svojstvo ne čuva podatke: ovim svojstvom se jednostavno pristupa drugim svojstvima i obezbeđuje virtuelna reprezentacija podataka:

```
type
  TMdListBoxDialog = class (TComponent)
  private
    FLines: TStrings;
    FSelected: Integer;
    FTitle: string;
    function GetSelItem: string;
    procedure SetLine (Value: TStrings);
    function GetLines: TStrings;
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
    function Execute: Boolean;
    property SelItem: string read GetSelItem;
  published
    property Lines: TStrings read GetLines write SetLines;
    property Selected: Integer read FSelected write FSelected;
    property Title: string read FTitle write FTitle;
end;
```

Veći deo koda primera se nalazi u metodu `Execute`, funkciji koja daje vrednost `True` ili `False` u zavisnosti od nepromenljivih rezultata okvira za dijalog. Ovo je standard za metod `Execute` većine standardnih Delphi komponenata okvira za dijalog. Funkcija `Execute` dinamički kreira formular, određuje neke od njegovih vrednosti koristeći svojstva komponente, prikazuje okvir za dijalog i, ukoliko je rezultat korektan, ažurira aktuelnu selekciju.

```
function TMdListBoxDialog.Execute: Boolean;
var
  ListBoxForm: TListBoxForm;
begin
  if FLines.Count = 0 then
    raise EStringListError.Create ('No items in the list');
  ListBoxForm := TListBoxForm.Create (Self);
  try
    ListBoxForm.ListBox1.Items: FLines;
    ListBoxForm.ListBox1.ItemIndex: FSelected;
    ListBoxForm.Caption := FTitle;
    if ListBoxForm.ShowModal = mr0k then
    begin
      Result := True;
      Selected := ListBoxForm.ListBox1.ItemIndex;
    end
    else
      Result := False;
  finally
    ListBoxForm.Free;
  end;
end;
```

Primetićete da se kod nalazi unutar `try-finally` bloka, te ukoliko se dese greške prilikom izvršavanja i prikazivanja okvira za dijalog, formular će se svakako ukloniti. Ja sam, takođe, koristio izuzetke za poziv greške ukoliko je lista prazna kada korisnik pokrene okvir za dijalog. Upotreba izuzetaka je dobra tehnika. Ostali metodi komponente se prilično lako mogu razumeti. Konstruktor kreira listu stringova `FLines`, koja se uklanja destruktorom; metodi `GetLins` i `SetLines` rade sa celom listom stringova, a funkcija `GetSelItem` (prikazana ispod) daje tekst selektovanog elementa:

```
function TMdListBoxDialog.GetSelItem: string;
begin
  if (Selected >= 0) and (Selected < FLines.Count) then
    Result := FLines [Selected]
  else
    Result := ' ';
end;
```

Naravno, pošto ručno pišemo kod komponente i dodajemo ga izvornom kodu originalnog formulara, ne smemo zaboraviti da napišemo proceduru `Register`.

Upotreba nevizuelne komponente

Kada je komponenta spremna, morate da obezbedite bitmapu. Za nevizuelne komponente bitmape su veoma važne jer se ne koriste samo za Cpmponent Palette, već i kada komponentu smestite na formular. Pripremimo bitmapu, instalirajmo kmponentu i napišimo jednostavan projekat da bismo je testirali. Formular test programa sadrži kontrolu, polje za izmene i našu novu nevizeulnu komponentu, kao što možete videti na slici 13.11.

SLIKA 13.11 *Formular primera ListDialDemo sa novom nevizuelnom komponentom*

Sada možete da napišete nekoliko linija koda koji pripada `OnClick` događaju kontrole:

```
procedure TForm1.Button1Click (Sender: TObject);
begin
  // select the text of the edit,
  // if corresponding to one of the strings
  MdListDialog1.Selected :=
    MdListDialog1.Lines.IndexOf (Edit1.Text);
  // run the dialog and get the result
  if MdListDialog1.Execute then
    Edit1.Text := MdListDialog1.SelItem;
end;
```

To je sve što je potrebno da pokrenete okvir za dijalog koji smo smestili u komponentu, što se može videti na slici 13.12. Kao što ste videli, ovo je interesantan pristup razvoju okvira za dijalog.

SLIKA 13.12 *Primer ListDialDemo prikazuje okvir za dijalog koji sam enkapsulirao u komponentu TListDial*

Definisanje korisničkih akcija

Pored definisanja korisničkih komponenata, možete definisati i registrovati nove standardne akcije koje će biti dostupne u Action Editoru komponente ActionList. Kreiranje novih akcija nije komplikovano. Potrebno je da izvršite nasleđivanje iz klase `TAction` i da zaobiđete neke metode osnovne klase.

U osnovi postoje tri metoda koja treba da zaobiđete. Funkcija `HandlesTarget` nas obaveštava o tome da li objekat akcije želi da obradi operaciju za aktuelni cilj, što je po definiciji kontrola koja ima fokus. Procedura `UpdateTarget` može da odredi korisnički interfejs kontrola koje su povezane sa akcijom, i po potrebi onemogući akciju ukoliko operacija trenutno ne može da se izvede. Konačno, možete implementirati metod `ExecuteTarget` da biste odredili kod koji treba izvršiti, tako da korisnik može da odabere akciju a da ne mora da je implementira.

Da bih Vam praktično pokazao ovaj pristup, implementirao sam tri cut, copy i paste akcije za listu, na način koji je sličan VCL načinu za polja za izmene (mada sam malo pojednostavio kod). Napisao sam osnovnu klasu koja sadrži korisnički kod, kojim se obrađuje status akcija, i tri izvedene klase kodom `ExecuteTarget`. Evo četiri klase:

```
type
  TMdListAction = class (TAction)
  public
    function HandlesTarget (Target: TObject): Boolean; override;
    procedure UpdateTarget (Target: Tobjectt override;
  end;

  TMdListCutAction = class (TMdListAction)
  public
    procedure ExecuteTarget(Target: TObject); override;
  end;

  TMdListCopyAction = class (TMdListAction)
  public
    procedure ExecuteTarget(Target: TObject); override;
  end;

  TMdListPasteAction = class (TMdListAction)
  public
    procedure UpdateTarget (Target: TObject); override;
    procedure ExecuteTarget (Target: TObject) override;
  end;
```

Metod `HandlesTarget` je implementiran samo za osnovnu klasu i aktivira akcije samo ukoliko je ciljna kontrola lista i ukoliko lista ima fokus:

```
function TMdListAction.HandlesTarget (Target: TObject) Boolean;
begin
  Result := (Target is TListBox) and
    TListBox(Target).Focused;
end;
```

Metod `UpdateTarget` ima dve različite implementacije. Unapred određena implementacija je u osnovnoj klasi i koristi se za akcije kopiranja i isecanja. Ove akcije su moguće samo ukoliko ciljna

lista sadrži bar jedan element i ukoliko je taj element trenutno selektovan. Status akcije Paste zavisi od statusa Clipboarda:

```
procedure TMdListAction.UpdateTarget (Target: TObject);
begin
  Enabled := ((Target as TListBox).Items.Count > 0) and
    ((Target as TListBox).ItemIndex >= 0);
end;

procedure TMdListPasteAction.UpdateTarget (Target: TObject);
begin
  Enabled := Clipboard.HasFormat (CF_TEXT);
end;
```

Konačno, tri metoda `ExecuteTarget` izvršavaju odgovarajuće akcije nad ciljnom listom:

```
procedure TMdListCopyAction.ExecuteTarget (Target: TObject);
begin
  with Target as TListBox do
    Clipboard.AsText := Items [ItemIndex];
end;

procedure TMdListCutAction.ExecuteTarget(Target: TObject);
begin
  with Target as TListBox do
  begin
    Clipboard.AsText := Items [ItemIndex];
    Items.Delete (ItemIndex);
  end;
end;

procedure TMdListPasteAction.ExecuteTarget(Target: TObject);
begin
  (Target as TListBox).Items.Add (Clipboard.AsText);
end;
```

Kada ste napisali ovaj kod u jedinici i kada ste ga dodali paketu (u ovom slučaju paketu MdPack), poslednji korak je registrovanje novih korisničkih akcija u datoj kategoriji. Ovo je naznačeno prvim parametrom procedure `RegisterActions`, dok se drugi odnosi na listu akcija klasa koje treba registrovati:

```
procedure Register;
begin
  RegisterActions ( 'ListBox',
    [TMdListCutAction, TMdListCopyAction, TMdListPasteAction],
    nil);
end;
```

Da bih testirao upotrebu ove tri akcije, ja sam napisao primer ListTest. Ovaj program sadrži dve liste i paletu alata sa tri kontrole povezane sa tri korisničke akcije i poljem za izmene kojim se unose nove vrednosti. Program korisniku omogućava isecanje, kopiranje i smeštanje elemenata liste. "Ništa naročito", pomislili ste, ali čudna je činjenica da program nema nikakav kod!

Pisanje editora svojstava

Pisanje komponenata je sasvim sigurno efektan način prilagođavanja Delphi okruženja i omogućava programerima da brže izrade aplikacije, a da ne moraju da imaju detaljno znanje o tehnikama niskog nivoa. Delphi okruženje je prilično otvoreno za proširivanje. Zapravo, lako je proširiti Object Inspector pisanjem editora svojstava i lako je proširiti Form Designer dodavanjem editora komponenata.

NAPOMENA

Pored ovih tehnika postoje brojni interni interfejsi koje Delphi nudi programerima alata. Upotreba ovih interfejsa zahteva dobro razumevanje funkcionisanja Delphi okruženja i prilično poznavanje mnogih naprednih tehnika. Pošto ove tehnike nisu potrebne svakom Delphi programeru, one se ne razmatraju u ovoj knjizi. Detaljan opis tradicionalnih Delphi interfejsa u vreme dizajniranja možete pronaći u knjizi "Delphi Developer's Handbook" (Sybex, 1998). Koliko ja znam, još uvek nema knjiga koje se detaljno bave ToolApijem koji je predstavljen u Delphiju 4. Tehničke informacije i neke primere ovih tehnika možete pronaći na mom web sajtu, kao i linkove ka drugim sajtovima gde su ove tehnike predstavljene. ■

Svaki editor svojstva mora biti potklasa apstraktne klase `TPropertyEditor`, koja je definisana u sistemskoj jedinici DsgnIntf (Design Interface). Ipak, Delphi već definiše neke specifične editore svojstava za stringove (klasu `TStringProperty`), celobrojne vrednosti (klasu `TIntegerProperty`), karaktere (klasu `TCharProperty`), kataloge (klasu `TEnumProperty`), skupove (klasu `TSetProperty`) i mnoge druge za boje, fontove, olovke, liste stringova i tako dalje.

U svakom korisničkom editoru svojstva morate ponovo definisati funkciju `GetAttributes` tako da daje skup vrednosti koje označavaju mogućnosti editora. Najvažniji atributi su `paValueList` i `paDialog`. Atribut paValueList označava da će Object Inspector prikazati combo polje sa spiskom vrednosti (koje mogu biti sortirane ukoliko je određen atribut `paSortList`) koje su obezbeđene zaobilaženjem metoda `GetValues`. Atribut `paDialog` aktivira kontrolu sa tri tačke u Object Inspectoru, koja izvršava metod Execute editora.

Editor za zvučna svojstva

Kontrola za zvuk koju smo ranije izradili sadržala je dva svojstva za zvuk: `SoundUp` i `SoundDown`. Ovo su, zapravo, bili stringovi, te smo mogli da ih prikažemo u Object Inspectoru upotrebom unapred određenog editora svojstva. Ipak, zahtevanje da korisnik unese naziv sistemskog zvuka ili eksternog fajla nije naročito ljubazno i podložno je greškama.

Kada je potrebno da odaberete fajl za string svojstvo, možete ponovo da upotrebite postojeći editor svojstva, klasu `TMPFilenameProperty`. Sve što je potrebno da učinite jeste da registrujete editor za svojstvo upotrebom specijalne procedure `RegisterPropertyEditor`:

```
RegisterPropertyEditor (
  TypeInfo (string), TDdhSoundButton,
  'SoundUp', TMPFileNameProperty);
```

Ovaj editor Vam omogućava da odaberete fajl za zvuk, ali želimo da imamo mogućnost da odaberemo i naziv sistemskog zvuka. (Kao što je ranije opisano, sistemski zvuci su unapred određeni nazivi zvuka koji su povezani sa korisničkim operacijama, a odnose na zvučne fajlove na strani Sound Windows Control Panela.) Zbog toga, umesto upotrebe ovog jednostavnog

pristupa, ja sam izradio komplikovaniji editor svojstva. Moj editor za stringove zvuka omogućava korisniku da odabere vrednost iz liste ili da prikaže okvir za dijalog iz koga se može učitati i testirati zvuk (zvuk iz fajla ili sistemski zvuk). Editor svojstva sadrži i metode `Edit` i `GetValues`.

```
type
  TSoundProperty = class (TStringProperty)
  public
    function GetAttributes: TPropertyAttributes; override;
    procedure GetValues (Proc: TGetStrProc); override;
    procedure Edit; overridde;
end;
```

SAVET

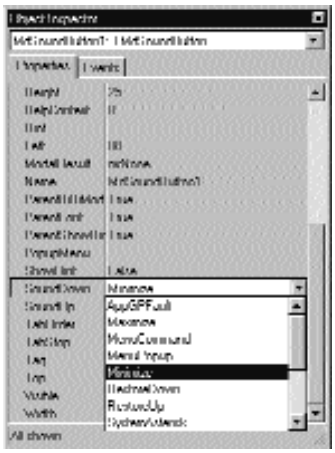
Konvencija Delphija za imenovanje klase editora svojstva zahteva da se naziv završava rečju Property, a svi editori komponenata završavaju rečju Editor. ■

Funkcija `GetAttributes` kombinuje atribute `paValueList` (za listu) i `paDialog` (za polje za izmene), a takođe sortira liste i omogućava selekciju svojstva za više komponenata:

```
function TSoundProperty.GetAttributes:
  TPropertyAttributes;
begin
  // editor, sorted list, multiple selection
  Result := [paDialog, paMultiSelect,
    paValueList, paSortList];
end;
```

Metod `GetValue` poziva proceduru koju dobija kao parametar mnogo puta, po jednom za svaki string koji želi da doda listi (kao što možete videti na slici 13.13):

```
procedure TSoundProperty.GetValues (Proc: TGetStrProc);
begin
  // provide a list of system sounds
  Proc ('Maximize');
  Proc ('Minimize');
  Proc ('MenuCommand');
  Proc ('MenuPopup');
  Proc ('RestoreDown');
  Proc ('RestoreUp');
  Proc ('SystemAsterisk');
  Proc ('SystemDefault');
  Proc ('SystemExclamation');
  Proc ('SystemExit');
  Proc ('SystemHand');
  Proc ('SystemQuestion');
  Proc ('SystemStart');
  Proc ('AppGPFault');
end;
```



SLIKA 13.13 *Lista zvukova pomaže korisniku, koji takođe može uneti vrednost svojstva ili dva puta kliknuti da bi aktivirao editor (koji je prikazan kasnije na slici 13.14)*

SAVET

Bolji pristup bi bio izdvajanje ovih vrednosti iz Windows Registryja gde su svi ovi nazivi izlistani. ■

NAPOMENA

Ukolko dalje želite da proširite ovaj primer, Delphi 5 Vam omogućava da dodate grafike listi koja se prikazuje u Object Inspectoru — ukoliko odlučite koje grafike da pridružite zvucima. ■

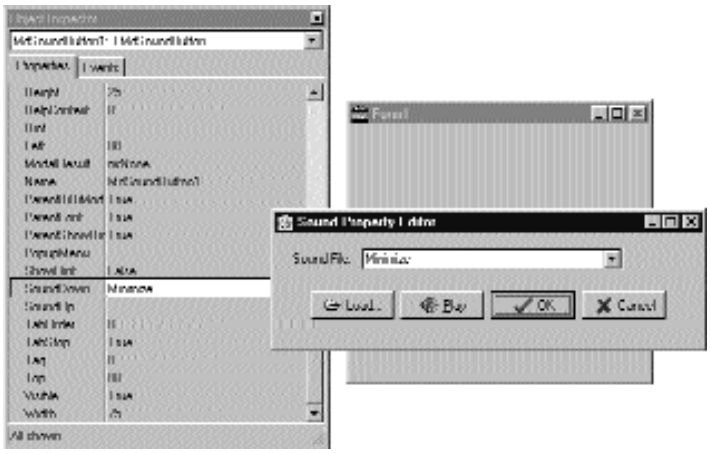
Metod `Edit` je veoma direktan jer samo kreira i prikazuje okvir za dijalog. Primetićete da smo mogli samo da prikažemo okvir za dijalog Open, ali smo odlučili da dodamo međukorak da bismo korisniku omogućili da testira zvuk. Ovo nalikuje postupku koji Delphi izvodi za grafička svojstva. Prvo se otvori pregled, a fajl se učitava samo ukoliko potvrdite da je korektan. Najvažniji korak je učitavanje fajla i njegovo testiranje pre nego što ga upotrebite za svojstvo. Evo koda metoda `Edit`:

```
procedure TSoundProperty.Edit;
begin
  SoundForm := TSoundForm.Create (Application);
  try
    SoundForm.ComboBox1.Text := GetValue;
    // show the dialog box
    if SoundForm.ShowModal = mrOK then
      SetValue (SoundForm.ComboBox1.Text);
  finally
    SoundForm.Free;
  end;
end;
```

Metodi `GetValue` i `SetValue`, koji se pozivaju u prethodnom kodu, definisani su u osnovnoj klasi string editorom svojstva. Oni jednostavno čitaju i zapisuju vrednost aktuelnog svojstva

komponente koju editujemo. Alternativno, komponenti koju editujete možete pristupiti upotrebom metoda `GetComponent` (koji zahteva parametar koji označava sa kojom selektovanom komponentom radite — 0 označava prvu komponentu). Kada komponenti direktno pristupate, takođe je potrebno da pozovete metod `Modified` objekta `Designer` (svojstvo editora svojstva osnovne klase). Poziv metoda `Modified` nam nije potreban za ovaj primer jer metod `SetValue` osnovne klase to automatski obavlja za nas.

Prethodni metod `Edit` prikazuje okvir za dijalog, standardni Delphi formular koji je vizuelno izrađen, i koji je dodat paketu koji sadrži komponente koje se koriste za vreme dizajniranja. Formular je prilično jednostavan; ComboBox prikazuje vrednosti koje dobija od metoda `GetValues`, a četiri kontrole Vam omogućavaju da otvorite fajl, testirate zvuk i da uklonite okvir za dijalog tako što ćete prihvatiti ili odbiti vrednosti. Primer okvira za dijalog možete videti na slici 13.14. Obezbeđivanje liste vrednosti i okvira za dijalog za izmenu svojstva prouzrokuje da Object Inspector prikazuje samo kontrolu sa strelicom koja označava listu, a ne prikazuje se kontrola sa tri tačke da bi se naznačilo da je na raspolaganju i editor okvira za dijalog.



SLIKA 13.14 *Formular editora svojstva zvuka prikazuje listu raspoloživih zvukova i omogućava Vam da učitate fajl i čujete odabrani zvuk*

Prve dve kontrole formulara sadrže po jednostavan metod pridružen njihovim `OnClick` događajima:

```
procedure TSoundForm.btnLoadClick (Sender: TObject);
begin
  if OpenDialog1.Execute then
    ComboBox1.Text := OpenDialog1.Filename;
end;

procedure TSoundForm.btnPlayClick (Sender: TObject);
begin
  PlaySound (PChar (ComboBox1.Text), 0, snd_Async);
end;
```

Na nesreću, pronašao sam jednostavan način da odredim da li je zvuk pravilno definisan i da li je raspoloživ (proverom fajla ukoliko je moguće, ali sistemski zvuci povlače nešto drugo). Funkcija `PlaySound` daje kod greške kada se zvuk pušta sinhrono, ali samo ukoliko ne može da pronađe unapred određeni sistemski zvuk koji pokušava da pusti ako ne može da pronađe zvuk koji ste zahtevali. Ukoliko zahtevani zvuk nije dostupan, pušta se unapred određeni sistemski zvuk i ne daje se kod greške. Funkcija `PlaySound` prvo pretražuje Registry i ukoliko ne pronađe zvuk, proverava da li naznačeni zvučni fajl postoji.

Instaliranje editora svojstva

Kada ste napisali ovaj kod, u Delphiju možete instalirati komponentu i njen editor svojstva. Da biste ovo obavili, potrebno je da dodate sledeći iskaz proceduri `Register` jedinice:

```
procedure Register;
begin
  ReisterPropertyEditor (TypeInfo (string),
    TMdSoundButton, 'SoundUp', TSoundProperty);
  ReisterPropertyEditor (TypeInfo (string),
    TMdSoundButton, 'SoundDown', TSoundProperty);
end;
```

Ovaj poziv registruje editor naznačen poslednjim parametrom za upotrebu uz svojstva tipa `string` (prvi parametar), ali samo za određenu komponentu i za svojstvo sa određenim nazivom. Poslednje dve vrednosti se mogu izostaviti da bi se obezbedili opštiji editori. Registrovanje ovog editora omogućava Object Inspectoru da prikaže listu vrednosti i okvir za dijalog koji se poziva metodom `Edit`.

Da bismo instalirali ovu komponentu, možemo dodati njen fajl sa izvornim kodom postojećem paketu ili novom paketu. Umesto dodavanja ove jedinice i drugih jedinica ovog poglavlja paketu MdPack, ja sam izradio drugi paket koji sadrži sve dodatke koji su izrađeni u ovom poglavlju. Paket je nazvan `MdDesPk` (što je skraćenica za "Mastering Delphi design package"). Ono što je novo a tiče se ovog paketa, jeste to da sam ga ja kompajlirao upotrebom direktive kompajlera `{$DESIGNONLT}`. Ova direktiva se koristi da bi se označili paketi koji imaju interakciju sa Delphi okruženjem, koji instalirajui komponente i editore, ali koji nisu potrebni u vreme izvršavanja aplikacija koje izrađujete.

NAPOMENA

Izvorni kod svih alata se nalazi u poddirektorijumu MdDesPk, kao i kod paketa koji se koristi za njihovo instaliranje. Nema primera koji pokazuju kako se koriste ovi alati, jer sve što je potrebno da učinite jeste da selektujete odgovarajuće komponente u Delphi okruženju i da vidite kako se ponašaju. ■

Jedinica editora svojstva koristi jedinicu SoundB, koja definiše komponentu `TMdSoundButton`. Zbog toga se novi paket referiše na postojeći paket. Evo početnog koda (ostale jedinice ću dodati kasnije u ovom poglavlju):

```
package MdDeskPk;

{$R *.RES}
{$ALIGN ON}
...
```

```
(*$DESCRIPTION 'Mastering Delphi DesignTime Package'}
{$DESIGNONLY}

requires
  vcl50;
  Mdpack;

contains
  PeSound in 'PeSound.pas'
  PeFSound in 'PeSound.pas' {SoundForm};
```

Pisanje editora komponente

Upotreba editora svojstva omogućava programerima da komponente učine pristupačnijim za korisnike. Zapravo, Object Inspector predstavlja jedan od ključnih delova korisničkog interfejsa Delphi okruženja i Delphi programeri ga često koriste. Ipak, postoji i drugi pristup koji možete prihvatiti da biste prilagodili interakciju komponente sa Delphijem: napišite editor komponente.

Baš kao što editor svojstva proširuje Object Inspector, editori komponenata proširuju Form Designer. Zapravo, kada kliknete desnim tasterom miša formular u vreme dizajniranja, prikazuju se neki unapred određeni elementi menija, kao i elementi koje dodaje editor komponente selektovane komponente. Primeri ovih elemenata menija su oni koji se koriste za aktiviranje Menu Designera, Fields Editora, Visual Query Buildera i drugih editora okruženja. Ponekad prikazivanje ovih specijalnih editora postaje unapred određena akcija komponente kada je dva puta kliknete.

Uobičajena upotreba editora komponente uključuje dodavanje polja About sa informacijama o programeru komponente, dodavanju naziva komponente i obezbeđivanju čarobnjaka za podešavanje svojstava.

Pravljenje potklasa klase TComponentEditor

Editor komponente treba da nasleđuje od klase `TComponentEditor`. Ova klasa sadrži četiri virtuelna metoda koja možete zaobići (i nekoliko manje važnih metoda koje ovde nećemo pominjati):

- Metod `GetVerbCount` daje broj elemenata menija koji se dodaju lokalnom meniju Form Designera kada se komponenta selektuje.
- Metod `GetVerb` se poziva po jednom za svaki od novih elemenata menija i trebalo bi da daje tekst koji treba da bude u lokalnom meniju za svaki od elemenata.
- Metod `ExecuteVerb` se poziva kada je jedan od elemenata menija selektovan. Broj elementa se prosleđuje kao parametar metoda.
- Metod `Edit` se poziva kada korisnik dva puta klikne komponentu u Form Designeru da bi aktivirao unapred određenu ikonu.
- Metod `PrepareItem` se poziva za svaku novu reč i omogućava Vam da izmenite pridruženi element menija (na primer, možete da definišete podelemente, dodelite bitmapu ili odredite oznaku). Ovo je novi metod u Delphiju 5.

Kada se naviknete na ideju da reč samo predstavlja novi element menija kojem je pridružena odgovarajuća akcija koju treba izvršiti, nazivi metoda ovog interfejsa postaju intuitivni. Ovaj interfejs je, zapravo, mnogo jednostavniji od editora svojstava koje smo ranije videli.

Editor komponente za ListDialog

Sada kada smo predstavili ključne ideje pisanja editora komponente, možemo se posvetiti primeru, editoru komponente ListDialog koju smo ranije izradili. U svom editoru komponente želeo sam da prikažem polje About, da meniju dodam informacije o pravima (nepravilna ali veoma česta upotreba editora komponente) i da omogućim korisnicima da izvrše specijalne akcije — da preliminarno prikažu okvir za dijalog koji je povezan sa komponentom. Takođe želim da promenim unapred određenu akciju da bih prikazao polje About posle zvučnog signala (što nije naročito korisno, ali pokazuje tehniku).

Da bi implementirao ovaj editor svojstva, program mora da zaobiđe četiri metoda koja su prethodno nabrojana:

```
uses
  DsgnIntf;

type
  TMdListCompEditor = class (TComponentEditor)
    function GetVerbCount: Integer; override;
    function GetVerb (Index: Integer): string; override;
    procedure ExecuteVerb (Index: Integer); override;
    procedure Edit; override;
  end;
```

Prvi metod daje broj elemenata menija koje želim da dodam lokalnom meniju:

```
function TMdListCompEdit.GetVerbCount: Integer;
begin
  Result := 3;
end;
```

Ovaj metod se poziva samo jednom i to pre prikazivanja menija. Drugi metod se poziva po jednom za svaki element menija, dakle, u ovom slučaju se poziva tri puta:

```
function TMdListCompEditor.GetVerb (
  Index: Integer): string;
begin
  case Index of
    0: Result := 'MdTabbedList (©Cantü)'
    1: Result := '&About this component...'
    2: Result := '&Preview'
  end;
end;
```

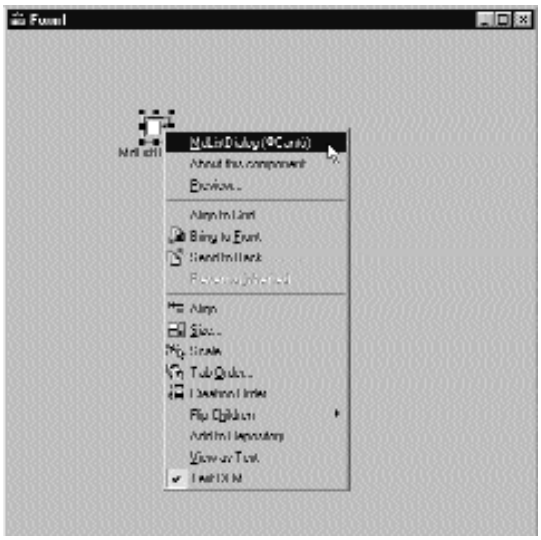
Efekat ovog koda je dodavanje elemenata menija lokalnom meniju formulara, kao što možete videti na slici 13.15. Selektovanjem bilo kog od elemenata menija aktivira se metod `ExecuteVerb` editora komponente:

```
procedure TMdListCompEditor.ExecuteVerb (
  Index:  Integer);
begin
  case Index of
    0..1: MessageDlg (
      'This is a simple component editor ' #13 +
      'built by Marco CantU' #13 +
      'for the book "Mastering Delphi"',
      mtInformation, [mbOK], 0);
    2: with Component as TMdListDialog do
      Execute;
  end;
end;
```

Ja sam odlučio da prva dva elementa obradim na jednoj grani iskaza **case**, mada sam mogao da preskočim kod elementa za informacije o pravima. Druga komanda menja pozive metoda `Execute` komponente koju editujemo, određujući upotrebu svojstva `Component` klase `TComponentEditor`. Pošto znamo tip komponente, lako možemo pristupiti njenim metodima posle dinamičke konverzije tipova.

Poslednji metod se odnosi na unapred određenu akciju komponente i aktivira se kada se komponenta dva puta klikne u Form Designeru:

```
procedure Edit;
begin
  // produce a beep and show the About box
  Beep;
  ExecuteVerb (0);
end;
```



SLIKA 13.15 *Elementi menija koji su dodati editorom svojstva komponente ListDialog*

Registrovanje editora komponente

Da bi ovaj editor bio dostupan Delphi okruženju, potrebno je da ga registrujemo. Još jednom možemo da dodamo proceduru `Register` njegovoj jedinici i pozovemo specifičnu registracionu proceduru za editore komponenata:

```
procedure Register;
begin
  RegisterComponentEditor (
    TMdListDialog, TMdListCompEditor);
end;
```

Ja sam ovu jedinicu dodao paketu MdDesPk, koji sadrži sva proširenja koja se koriste u vreme dizajniranja, koja smo izradili u ovom poglavlju. Posle instaliranja i aktiviranja ovog paketa možete kreirati novi projekat, postaviti komponentu liste i eksperimentisati njom.

Šta je sledeće?

U ovom poglavlju smo videli kako da definišemo različite tipove svojstava, kako da dodamo događaje i kako da definišemo i zaobiđemo metode komponente. Videli smo četiri različita primera komponenata, uključujući i jednostavne izmene postojećih komponenata, nove grafičke komponente i, u poslednjem delu, okvir za dijalog unutar komponente. Prilikom izrade ovih komponenata susreli smo se sa novim izazovima Windows programiranja. Uopšte uzev, programeri često imaju potrebu da direktno koriste Windows API prilikom pisanja novih Delphi komponenata.

Pisanje komponenata je veoma zgodna tehnika za ponovnu upotrebu softvera, ali da bi se Vaše komponente lakše upotrebljavale, potrebno je da pokušate da ih integrišete u Delphi okruženje koliko god je to moguće, tako što ćete pisati editore svojstava i editore komponenata.

Postoje mnoga druga proširenja Delphi IDE-a koja možete napisati, uključujući čarobnjake. Delphi, zapravo, sadrži priličan broj ToolsApija, koji su delimično dokumentovani u nekim knjigama, uključujući i moju knjigu *Delphi Developer's Handbook* (Sybex, 1998). Ja sam napisao veliki broj proširenja za Delphi, od kojih su neka dostupna na mom web sajtu, `www.marcocantu.com`.

Posle razmatranja komponenata i malog izleta u Delphi okruženje, naredno poglavlje se odnosi na Delphi DLL-ove. Njih smo već sretali u mnogim prošlim poglavljima i vreme je za detaljno razmatranje njihove uloge i njihove izrade. U istom poglavlju ću proširiti razmatranje upotrebe Delphi paketa, koji predstavljaju specijalni tip DLL-ova.

POGLAVLJE

14

Dinamičke biblioteke za povezivanje i paketi

Postoje dva tipa Windowsovih izvršnih fajlova: programi i dinamičke biblioteke za povezivanje (DLL-ovi). Kada pišete Delphi aplikacije, obično generišete fajl programa, dakle, EXE fajl. Ipak, Delphi aplikacije često koriste pozive funkcija koje se čuvaju u DLL-ovima. Delphi, takođe, omogućava programerima da koriste DLL u vreme izvršavanja za biblioteku komponenata. Kada kreirate paket, Vi u osnovi kreirate DLL. Delphi, takođe, može da generiše čiste dinamičke biblioteke za povezivanje. Strana New Object Repositoryja sadrži DLL generator, koji generiše veoma malo linija izvornog koda.

Veoma je jednostavno generisati DLL-ove u Delphi okruženju. Ipak, javljaju se neki problemi vezani za prirodu DLL-ova. Pisanje DLL-ova u Windowsu nije uvek jednostavno kao što se možda čini, jer se DLL i program koji vrši pozive moraju pridržavati konvencija poziva, tipova parametara i drugih detalja. Ovo poglavlje se odnosi na osnove DLL programiranja sa tačke gledišta Delphija, a u njemu ćete pronaći jednostavne primere onoga što možete smestiti u Delphi DLL. Prilikom razmatranja primera referisaću se i na druge programske jezike i okruženja jer je jedan od ključnih razloga pisanja procedura u DLL-u mogućnost da se procedure pozovu iz programa koji je napisan na nekom drugom jeziku.

Poslednji deo poglavlja je posvećen specijalnom tipu dinamičke biblioteke za povezivanje, Delphi *paketu*. Ove pakete nije lako koristiti kako se to na prvi pogled čini, i Delphi programerima je bilo potrebno prilično vremena da shvate kako da ih praktično podrže. Ja ću ovde sa Vama podeliti neke od ovih interesantnih saveta i tehnika.

Uloga DLL-ova pod Windowsom

Pre nego što se pozabavim programiranjem DLL-ova u Delphiju i drugim programskim jezicima, prikazaću kratak tehnički uvod u DLL-ove pod Windowsom, ističući ključne elemente. Počećemo proučavanjem dinamičkog povezivanja, zatim ćemo videti kako Windows koristi DLL-ove, razmatraćemo razlike između DLL-ova i izvršnih fajlova, a završićemo pravilima koja treba primeniti prilikom pisanja DLL-ova.

Šta je dinamičko povezivanje?

Pre svega, potrebno je da razumete razlike između statičkog i dinamičkog povezivanja funkcija ili procedura. Kada podrutine nisu direktno dostupne u izvornom fajlu, kompajler dodaje podrutine internoj tabeli, koja sadrži sve spoljašnje (eksterne) simbole. Naravno, kompajler mora da zna deklaraciju podrutina i parametre kao i njihov tip ili će prijaviti grešku.

Posle kompilacije normalne — *statičke* — podrutine, linker pronalazi kompajlirani kod podrutine iz Delphi kompajlirane jedinice (ili statičke biblioteke) i dodaje je izvršnom fajlu. Rezultujući EXE fajl sadrži sav kod programa i jedinice koje se koriste. Delphi linker je dovoljno pametan da uključi minimalnu količinu koda jedinica koje koristi program i da poveže samo funkcije i metode koji se zaista koriste.

NAPOMENA

Primetan izuzetak od ovog pravila su virtuelni metodi. Kompajler ne može unapred da odredi koje virtuelne metode će program pozivati, te ih stoga sve uključuje. Zbog toga, programi i biblioteke koje sadrže mnogo virtuelnih funkcija mogu da generišu veće izvršne fajlove. Prilikom razvoja VCL-a, Borlandovi programeri su morali da izbalansiraju fleksibilnost, koja se dobija virtuelnim funkcijama, i veličinu izvršnih fajlova, koja se dobija ograničenim brojem virtuelnih funkcija. ■

U slučaju dinamičkog povezivanja koje se dešava kada Vaš kod poziva funkcije iz DLL-ova, linker koristi samo informacije iz deklaracije `external` podrutine za podešavanje nekih tabela izvršnog fajla. Kada Windows učita izvršni fajl u memoriju, prvo učitava sve potrebne DLL-ove, a tek onda program počinje da se izvršava. Tokom procesa učitavanja, Windows popunjava interne tabele programa adresama funkcija iz DLL-ova koje se nalaze u memoriji. Ukoliko iz nekog razloga DLL ne može da se pronađe, program neće ni početi da se izvršava, često prikazujući besmislene greške (recimo, ozloglašenu poruku "a device attached to your system is not functioning" — uređaj koji je priključen na Vaš sistem ne funkcioniše).

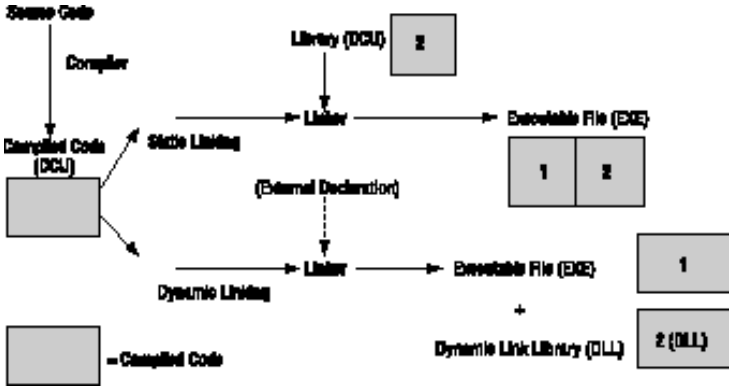
Svaki put kada program poziva spoljašnju funkciju, on koristi internu tabelu da poziv prosledi DLL kodu (koji se sada nalazi u adresnom prostoru programa). Primetićete da ovakva šema ne uključuje dve različite aplikacije. DLL postaje deo programa koji se izvršava i učitava se u isti adresni prostor. Sva prosleđivanja parametara se odvijaju u steku aplikacije (jer DLL-ovi nemaju zaseban stek).

Dijagram koji prikazuje kako programi pozivaju statički ili dinamički povezane funkcije, možete videti na slici 14.1. Primetili ste da do sada nisam razmatrao kompajliranje DLL-ova — to je zato što sam prvo želeo da se usredsredim na dva različita mehanizma povezivanja.

NAPOMENA

Kada se termin dinamičko povezivanje (dynamic linking) odnosi na DLL-ove, nema nikakve veze sa karakteristikom kasnog povezivanja objektno orijentisanih jezika. Virtuelni i dinamički metodi u Object Pascalu nemaju nikakve veze sa DLL-ovima. Na nesreću, isti termin se koristi za obe vrste procedura i funkcija, što stvara priličnu zabunu. Kada u ovom poglavlju govorim o dinamičkom povezivanju, to se ne odnosi na polimorfizam već na funkcije DLL-ova. ■

Postoji još jedan, manje uobičajen pristup upotrebe DLL-ova, koji je još dinamičniji od pristupa koji smo prethodno razmatrali. Zapravo, u vreme izvršavanja, Vi možete da učitate DLL u memoriju, potražite funkciju (pod uslovom da znate njen naziv) i pozovete funkciju po nazivu. Ovakav pristup zahteva složeniji kod i u opštem slučaju je sporiji. Pozitivna strana pristupa je to što ne morate da imate DLL na raspolaganju prilikom pokretanja programa. Ovaj pristup ćemo koristiti u primeru DynaCall kasnije u ovom poglavlju.



SLIKA 14.1 *Statičko i dinamičko povezivanje u Windowsu*

Čemu služe DLL-ovi?

Sada kada imate uopštenu ideju o funkcionisanju DLL-ova, možemo se pozabaviti razlozima njihove upotrebe u Windowsu.

- Ukoliko različiti programi koriste isti DLL, on se u memoriju učitava samo jednom, a time se štedi sistemska memorija. DLL-ovi se mapiraju u privatni adresni prostor svakog procesa (svake aplikacije koja se izvršava), ali se njihov kod samo jedanput učitava u memoriju.

NAPOMENA

Operativni sistem će pokušati da u memoriju učita DLL na istu adresu u svakom adresnom prostoru aplikacije, ali ukoliko ta adresa nije raspoloživa u određenom virtuelnom adresnom prostoru aplikacije, DLL slika koda za taj proces će morati da bude premeštena. Primetićete da se premeštanje obavlja za svaki proces, a ne na nivou sistema. ■

- Možete da obezbedite različite verzije DLL-ova zamenjujući postojeću verziju. Ukoliko podrutine DLL-a imaju iste parametre, možete da pokrenete program upotrebom nove verzije DLL-a, a da ne morate ponovo da ga kompajlirate. Ukoliko DLL sadrži nove podrutine, to nema nikakvu važnost. Problem se može pojaviti samo ukoliko rutina iz stare verzije DLL-a ne postoji u novoj verziji.

VERZIJE DLL PARAMETARA

Najnovije Windows API funkcije često kao jedan parametar koriste pokazivač na strukturu podataka, koja sadrži aktuelni parametar. Ovakav pristup omogućava kreatoru DLL-a da doda nove parametre strukturi podataka, a da to ne utiče na postojeći kod. Tipično, u ovim slučajevima, prvi parametar strukture podataka čuva njegovu veličinu, koja se koristi za naznačavanje verzije strukture. Na ovaj način DLL može da odredi na koju verziju strukture podataka se referiše aplikacija, samo proverom parametra veličina/verzija. Ovo je koristan pristup koji bi trebalo da sledite ukoliko smatrate da će se parametri funkcije gotovo sigurno promeniti u budućoj verziji DLL-a.

Za neke primere možete da pogledate Windows API funkcije za okvire za dijalog (kao što su `GetOpenFileName` ili `ChooseColor`) u Help fajlu, ali mnoge nove API funkcije koriste isti pristup.

Ove generičke prednosti se odnose na nekoliko slučajeva. Ukoliko imate složen algoritam, ili neke složene formulare koristite u nekoliko aplikacija, možete ih sačuvati u DLL-u. Ovo će Vam omogućiti da smanjite veličinu izvršnog fajla i sačuvate nešto memorije kada pokrenete nekoliko aplikacija koje istovremeno koriste te DLL-ove.

Druga prednost je naročito primenljiva na složene aplikacije. Ukoliko imate veoma veliki program koji zahteva česta ažuriranja i prepravke, deljenje programa u nekoliko izvršnih fajlova i DLL-ova Vam omogućava da distribuirate samo promenjene delove programa umesto da imate jedan veliki izvršni fajl. Ovo naročito ima smisla za Windows sistemske biblioteke. Ukoliko Borland (Inprise) napravi novu verziju Database Engine bibilioteka, ili napiše novi SQL Links za pristup drugim SQL serverima baza podataka, nećete ponovo morati da kompajlirate svoju aplikaciju da biste mogli da koristite izmene.

Druga uobičajena tehnika je da DLL-ove koristite samo za čuvanje resursa. Možete da izradite različite verzije DLL-ova koji sadrže stringove za različite jezike i da zatim promenite jezik za vreme izvršavanja, ili možete da pripremite biblioteku ikona i bitmapa i zatim ih koristite za različite aplikacije. Razvoj verzija koje zavise od jezika programa je naročito važan, a Delphi 5 ima podršku za ovakav razvoj kroz Integrated Translation Environment (ITE — integrisano okruženje za prevođenje), koje je opisano u Poglavlju 19.

Još jedna važna prednost je da DLL-ovi ne zavise od programskog jezika. Većina Windows programskih okruženja, uključujući i većinu makro jezika korisničkih aplikacija, omogućava programeru da

poziva podrutine koje se čuvaju u DLL-u. To znači da možete da izradite DLL u Delphiju i pozivate ga iz C++, Visual Basica, Excela, WordPerfecta i mnogih drugih Windows aplikacija.

Razumevanje sistemskih DLL-ova

Windowsovi sistemski DLL-ovi koriste sve ključne prednosti DLL-ova koje sam do sada istakao. Zbog toga ih vredi proučiti. Prvo, Windows sadrži mnogo sistemskih DLL-ova. Tri centralna dela Windowsa — Kernel, User i GDI — su implementirana upotrebom DLL-ova.

U Windowsu 95 i Windowsu 98 tri ključne biblioteke duplirane su u 16-bitnim verzijama (`KRNL386.EXE`, `USER.EXE` i `GDI.EXE`) i 32-bitnim verzijama (`KERNEL32.DLL`, `USER32.DLL` i `GDI32.DLL`). Ove dve verzije često pozivaju jedna drugu u procesu koji se naziva *thunking*. U Windowsu NT (i Windowsu 2000) sistemske biblioteke imaju samo 32-bitni kod. Ostali sistemski DLL-ovi su proširenja operativnog sistema, drajveri uređaja, fontovi, ActiveX kontrole i mnoga druga proširenja.

Kada je u pitanju sam Windows, upotreba DLL-ova je veoma važna. Zapravo, DLL-ovi su jedna od ključnih osnova operativnog sistema Windows. Pošto svaka aplikacija koristi sistemske DLL-ove za sve, od kreiranja prozora do prikazivanja izlaza, svaki program je povezan sa tim DLL-ovima. Posvetimo ovome malo pažnje da bismo videli zašto, možda, imamo različite verzije iste biblioteke.

Prvo, uzmimo u obzir drajvere uređaja. Kada promenite štampač, nemate potrebu da ponovo izradite svoju aplikaciju niti da kupite novu verziju Windows GDI biblioteke, koja manipuliše izlazom štampača. Potrebno je samo da obezbedite određeni drajver, a to je DLL koji GDI poziva da bi pristupio Vašem štampaču. Svaki tip štampača ima svoj DLL drajver, što sistem čini neverovatno fleksibilnim.

S druge tačke gledišta, administracija verzija je veoma važna za sam sistem. Ukoliko imate aplikaciju koja je kompajlirana za Windows 3.1, trebalo bi da možete da je pokrenete na bilo kojoj Win32 platformi. Svaka verzija Windowsa sadrži drugačiji sistemski kod (a Win32 16-bitna podrška zapravo ispravlja neke nedostatke Windowsa 3.1), ali kako svaka nova verzija sadrži starije API funkcije, stari kod i dalje funkcioniše, mada ne može da iskoristi nove API funkcije. Ipak, stari kod zaista može da iskoristi nove karakteristike kada se promeni kod postojeće funkcije. Očigledan primer je korisnički interfejs: ako izradite aplikaciju za Windows 3.1, možete da je pokrenete u Windowsu 95 i Windowsu 98, i ona će automatski imati drugačije elemente korisničkog interfejsa. Niste ponovo kompajlirali program; on je koristio karakteristike novih sistemskih biblioteka koje su sa programom dinamički povezane.

Sistemski DLL-ovi se, takođe, koriste kao arhive za sistemske informacije. Na primer, USER.DLL održava spisak svih aktivnih prozora na sistemu, dok GDI.DLL čuva listu aktivnih olovki, četkica, ikona, bitmapa i slično. Slobodna memorijska oblast ovih DLL-ova se obično naziva "free system resources" (slobodni sistemski resursi) i ima veoma važnu ulogu u Windowsu.

Razlike između DLL i EXE fajlova

Sada kada ste naučili osnovne elemente dinamičkog povezivanja i neke od razloga za upotrebu ove tehnike, možemo se okrenuti razlikama između normalnih izvršnih fajlova (EXE fajlova) i dinamičkih biblioteka za povezivanje (DLL fajlova). Interna struktura EXE fajla je većim delom identična strukturi DLL fajla. Stvari se menjaju tek kada se DLL učita u memoriju.

Kao što sam ranije istakao, Windows učitava kod DLL-a u memoriju samo jednom. Isto se dešava i kada je u pitanju izvršni fajl, čak i kada pokrenete više kopija. U oba slučaja mehanizam za brojanje upotrebe modula osigurava da se kod DLL-a izbaci iz memorije kada se zatvore svi programi koji koriste DLL.

Ključna razlika između programa i DLL-ova je u tome da DLL, čak i kada je učitan u memoriju, nije program koji se izvršava. DLL je samo kolekcija procedura i funkcija koje pozivaju drugi programi. Ove procedure i funkcije koriste stek programa koji ih poziva (da budemo precizni, pozivnu vezu — *calling thread*). Dakle, druga ključna razlika između programa i biblioteka je u tome da biblioteke ne kreiraju sopstveni stek — biblioteke koriste stek programa koji ih poziva. Pod platformom Win32, bilo kakva alokacija memorije od strane DLL-a ili bilo koji globalni podatak koji DLL kreira, smešta se u adresni prostor glavnog procesa pošto su DLL-ovi učitani u adresni prostor aplikacije.

Pravila za Delphi programere DLL-ova

Sve što sam do sada opisao može se rezimirati u nekoliko pravila za DLL programere. DLL funkcija ili procedura koju će pozivati spoljašnji program mora da sledi sledeća pravila:

- Mora biti izlistana u DLL-ovoj klauzuli `exports`. Ovo rutinu čini dostupnom spoljašnjem svetu.
- Izvezene funkcije takođe moraju biti deklarisane kao `stdcall` da bi koristile standardnu tehniku prosleđivanja parametara za Win32 umesto optimizovane tehnike prosleđivanja parametara `register` (što je unapred određena tehnika za Delphi).
- Tipovi parametara DLL-a bi trebalo da budu unapred određeni Windowsovi tipovi, bar ukoliko želite da imate mogućnost da DLL koristite u okviru drugih razvojnih okruženja. Postoje i druga pravila za izvoženje stringova, što ćemo videti u primeru FirstDll.
- DLL može da koristi globalne podatke koji se ne dele sa aplikacijom iz koje se vrši poziv. Svaki put kada aplikacija učita DLL, ona čuva globalne podatke DLL-a u svom adresnom prostoru, što ćemo videti u primeru DllMem.

Win16 i Win32 DLL-ovi

Drugi važan aspekt DLL-ova je da se u Windowsu javljaju u dva različita oblika. Postoje Windows 3.1 (Win16) DLL-ovi i Windows NT, Windows 95 ili Windows 98 (Win32) DLL-ovi. Biblioteke pisane 16-bitnom verzijom Delphija su prva vrsta DLL-ova. Biblioteke kompajlirane 32-bitnom verzijom Delphija su druga vrsta DLL-ova.

Na nesreću, što sam već istakao, 16-bitne i 32-bitne verzije DLL-ova *nisu* kompatibilne. Na primer, ne možete pozvati 16-bitnu verziju iz 32-bitnog Delphi programa. Ovo nije ograničenje vezano za Delphi, već je to opšti problem kod Windowsa. Zapravo, postoji i rešenje: možete da koristite Microsoftov thunk kompajler za kreiranje pravilnih ulaznih tačaka za različite tipove DLL-ova. To je ono što Windows 95 i Windows 98 čine prilikom pozivanja 16-bitnih sistemskih biblioteka iz 32-bitnih aplikacija, ili prilikom poziva novih 32-bitnih sistemskih biblioteka iz starih 16-bitnih aplikacija.

Ipak, upotreba thunk mehanizma je prilično složena i omogućava Vašim 32-bitnim aplikacijama da se izvršavaju samo u Windowsu 95 ili Windowsu 98, ali ne i u Windowsu NT. U većini slučajeva rešenje ovog problema je kreiranje 16-bitne aplikacije koja pristupa 16-bitnim DLL-ovima, a zatim upotreba jedne od tehnika koje su Vam na raspolaganju (memorijski mapirani fajlovi ili `WM_COPYDATA` poruke) da biste 16-bitnoj aplikaciji omogućili da deli podatke sa 32-bitnom verzijom Vašeg programa. Thunking se veoma sporo izvršava jer zahteva promenu modova kernela.

Mada su osnovni delovi Windowsa 98 još uvek načinjeni od 16-bitnih DLL-ova, ovaj 16-bitni svet polako izumire. Zbog toga je pozivanje 16-bitnih DLL-ova iz Vaših 32-bitnih Delphi programa sve ređe i ređe.

Upotreba postojećih DLL-ova

Mi smo već koristili postojeće DLL-ove u mnogim primerima ove knjige prilikom pozivanja Windows API funkcija. Kao što se možda sećate, sve API funkcije su deklarisane u sistemskoj Windows jedinici. Funkcije su deklarisane u `interface` odeljku jedinice, kao što je ovde pokazano:

```
function PlayMetaFile (DC: HDC; MF: HMETAFILE): BOOL; stdcall;
function PaintRgn (DC: HDC; RGN: HRGN): BOOL; stdcall;
function PolyPolygon (DC: HDC; var Points; var nPoints;
  p4: Integer): BOOL; stdcall;
function PtInRegion (RGN: HRGN; p2, p3: Integer): BOOL; stdcall;
```

Zatim, u odeljku `implementation`, umesto da se obezbedi kod svake funkcije, jedinica se referiše na spoljašnju definiciju u DLL-u:

```
const
  gdi32 = 'gdi32.dll';

function PlayMetaFile; external gdi32 name 'PlayMetaFile';
function PaintRgn; external gdi32 name 'PaintRgn';
function PolyPolygon; external gdi32 name 'PolyPolygon';
function PtInRegion; external gdi32 name 'PtInRegion';
```

NAPOMENA

U fajlu Windows.PAS dosta se koristi direktiva `{$EXTERNALSYM identifierć`. Ovo ima malo veze sa samim Delphijem; to se odnosi na C++ Builder. Ovaj simbol sprečava da se odgovarajući Pascal simbol pojavi u C++ prevedenom heder fajlu. Ovim se sinhronizuju Delphi i C++ identifikatori, tako da se kod može deliti između dva jezika. ■

Spoljašnja definicija ovih funkcija se odnosi na naziv DLL-a koji koriste. Naziv DLL-a mora da sadrži DLL ekstenziju, inače će program funkcionisati pod Windowsom 95 i Windowsom 98, ali ne i pod Windowsom NT. Drugi element je naziv same funkcije DLL-a. Direktiva name nije neophodna ukoliko se naziv Pascal funkcije (ili procedure) odgovara nazivu funkcije DLL-a (razlikuju se mala i velika slova).

Da biste pozvali funkciju koja se nalazi u DLL-u, možete da obezbedite njenu deklaraciju i spoljašnju definiciju, kao što smo malopre pokazali, ili možete da ih spojite u jednu deklaraciju. Kada je funkcija pravilno definisana, možete je pozivati iz koda Vaše Delphi aplikacije kao i bilo koju drugu funkciju. Nema ničeg specijanog u sintaksi poziva; to je samo normalan poziv procedure ili funkcije.

Kao primer, ja sam napisao veoma jednostavan DLL u jeziku C++, koji sadrži neke trivijalne funkcije, samo da bih Vam pokazao kako da pozivate DLL-ove iz Delphi aplikacija. Neću detaljno objašnjavati C++ kod (inače, to je u osnovi C kod), već ću se pozabaviti pozivima između Delphi aplikacija i C++ DLL-a. Za Delphi programiranje je veoma uobičajeno koristiti DLL-ove napisane u jeziku C i C++.

NAPOMENA

Pojavom Borland C++ Buildera (Delphi klon zasnovan na jeziku C++) mogućnosti deljenja koda između C++ i Object Pascal aplikacija su se eksponencijalno povećale. C++ Builder može direktno da čita Pascal jedinice i koristi Delphi komponente. Ono što ovde razmatram više je generički i tradicionalniji pristup. ■

Upotreba C++ DLL-a

Pretpostavimo da ste dobili DLL izrađen u jeziku C i C++. Uopšte uzev, u rukama ćete imati .DLL fajl (samo kompajliranu biblioteku), .H fajl (deklaraciju funkcija koje se nalaze u biblioteci) i .LIB fajl (još jednu verziju liste izvezenih funkcija za C/C++ linker). Ovaj .LIB fajl je potpuno beskoristan u Delphiju, .DLL fajl se koristi kao takav, a .H fajl treba prevesti u Pascal jedinicu sa odgovarajućim deklaracijama.

U narednom listingu možete videti deklaracije C++ funkcija koje sam koristio za izradu CppDll biblioteke. Kompletan izvorni kod i kompajlirana verzija C++ DLL-a i Delphi aplikacije koja koristi DLL nalaze se u direktorijumu CppDll, među direktorijumima koje ste preuzeli. Trebalo bi da možete da kompajlirate ovaj kod upotrebom bilo kog C++ kompajlera; ja sam ga testirao samo sa Borlandovim kompajlerima (Borland C++ 5.0 i C++ Builder 3 i 4). Evo C++ deklaracija funkcija:

```
extern "C" __declspec (dllexport)
int WINAPI Double (int n);
extern "C" __declspec (dllexport)
int WINAPI Triple (int n);
__declspec (dllexport)
int WINAPI Add (int a, int b);
```

Tri funkcije obavljaju neke osnovne računske operacije nad parametrima i daju rezultat. Primetićete da su sve funkcije definisane `WINAPI` modifikatorom, koji određuje pravilnu konvenciju poziva parametara, a prethodi im deklaracija `__declspec (dllexport)`, koja čini funkcije dostupnim spoljašnjem svetu.

Dve od ovih C++ funkcija koriste konvencije imenovanja C-a (naznačene iskazom extern "C"), ali treća, `Add`, to ne čini. Ovo utiče na način na koji pozivamo ove funkcije u Delphiju. Zapravo, interni nazivi ove tri funkcije odgovaraju njihovim nazivima u C++ fajlu sa izvornim kodom, izuzev funkcije `Add`. Pošto nismo koristili klauzulu `extern "C"` za ovu funkciju, C++ kompajler je koristio izdvajanje naziva (name mangling). Ova tehnika se koristi za uključivanje informacija o broju i tipu parametara u nazivu funkcije, koje C++ zahteva da bi mogla da se implementira zamena funkcija. Rezultat prilikom upotrebe Borland C++ kompajlera je smešan naziv funkcije: `@add$qqsii`. Ovo je, zapravo, naziv koji treba da koristimo u našem Delphi primeru da bismo pozvali `Add` DLL funkciju (što objašnjava zašto treba izbegavati C++ izdvajanje naziva u izvezenim funkcijama, i zbog čega ih uopšte deklarišemo kao `extern "C"`). Slede deklaracije tri funkcije u Delphi CallCpp primeru:

```
function Add (A, B: Integer): Integer;
  stdcall; external 'CPPDLL.DLL' name '@add$qqsii';
function Double (N: Integer): Integer;
  stdcall; external 'CPPDLL.DLL' name 'Double';
function Triple (N: Integer): Integer;
  stdcall; external 'CPPDLL.DLL';
```

Kao što vidite, možete da obezbedite ili izostavite alijas za spoljašnju funkciju. Ja sam jedan alijas dodelio prvoj funkciji (nije bilo alternative, jer izvezeni naziv funkcije `@add$qqsii` nije identifikator koji može da se koristi u Pascalu) i drugoj funkciji, mada u drugom slučaju to nije bilo neophodno. Ukoliko se dva naziva podudare, možete da izostavite direktivu name, kao što sam ja to učinio za treću funkciju.

Ne zaboravite da dodate direktivu `stdcall` svakoj definiciji, tako da modul koji poziva (aplikacija) i modul koji se poziva (DLL) koriste istu konvenciju prosleđivanja parametara. Ukoliko to ne učinite, kao parametri će se prosleđivati slučajne vrednosti, što je greška koju je veoma teško pronaći.

NAPOMENA

Kada je potrebno da konvertujete veliki C/C++ heder fajl u odgovarajuće Pascal deklaracije, umesto da vršite ručnu konverziju, možete da upotrebite alat da biste delimično automatizovali proces. Jedan od takvih alata je HeadConv, koji je napisao Bob Svort (Bob Swart). Kopiju alata možete pronaći na njegovom web sajtu `http://www.drbob42.com`. ■

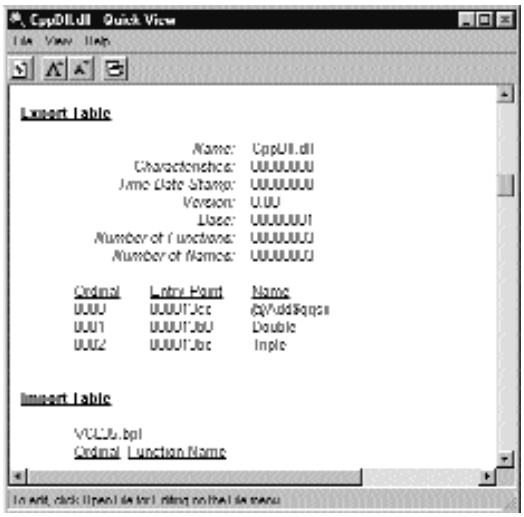
Ukoliko niste sigurni za nazive funkcija koje su izvezene DLL-om, možete jednostavno da selektujete DLL fajl u Windows Exploreru, kliknete fajl desnim tasterom miša i odaberete komandu QuickView. Program koji se pojavljuje prikazuje neke tehničke informacije niskog nivoa koje su dostupne za svaki izvršni fajl. Ono što nas sada zanima je sekcija *Export Table*, kao što je prikazano na slici 14.2.

Primetićete da svaka od tri funkcije ima naziv i broj indeksa (označen kao Ordinal). Broj indeksa se generalno koristi za povezivanje DLL funkcija pod 16-bitnim Windowsom. Pod Win32 Microsoft sugeriše da DLL funkcije povezujete preko naziva.

Alternativa programu QuickView je upotreba programa TDump32 koji dobijate uz Delphi, koji će Vam dati još više detalja o internoj strukturi izvršnog fajla. Imajte na umu da programi QuickView i Tdump32 mogu da se koriste kako za izvršne fajlove, tako i DLL-ove.

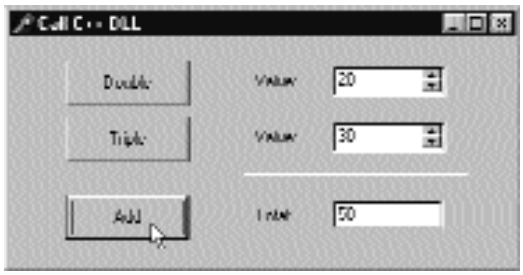
Da bih upotrebio ovaj C++ DLL, ja sam izradio Delphi primer koji sam nazvao CallCpp. To je jednostavan formular koji sadrži tri kontrole za poziv po jedne od funkcija DLL-a, dve komponente SpinEdit za parametre i polje za izmene koje se može samo čitati za prikazivanje rezultata. Ukoliko klikente prvu kontrolu, vrednost odgovarajuće komponente SpinEdit se udvostručava:

```
procedure TForm1.BtnDoubleClick (Sender: TObject);
begin
  SpinEdit1.Value := Double (SpinEdit1.Value);
end;
```



SLIKA 14.2 *Windows QuickView Vam omogućava da pretražite DLL i EXE fajl. Ovde je prikazan Export Table fajla CPPDLL.DLL.*

Kod za kontrolu Triple je veoma sličan. Kada kliknete treću kontrolu, Add, program sabira - vrednosti dve komponente SpinEdit pozivom treće funkcije DLL-a, a rezultat prikazuje u polju za izmene. Na slici 14.3 je prikazan primer izlaza kada je svaka kontrola po jednom kliknuta.



SLIKA 14.3 *Izlaz primera CallCpp kada ste kliknuli svaku od kontrola*

Evo koda obrade događaja `OnClick` za treću kontrolu:

```
procedure TForm1.BtnAddClick (Sender: TObject);
begin
  Edit1.Text := IntToStr ( Add (
    SpinEdit1.Value, SpinEdit2.Value));
end;
```

Da biste pokrenuli aplikaciju, DLL treba da se nalazi u istom direktorijumu kao i projekat, u jednom od direktorijuma navedenih u putanji, ili u Windows ili System direktorijumima. Ukoliko izvršni fajl premestite u novi direktorijum i pokušate da ga pokrenete, javiće se greška kojom se navodi da DLL nedostaje, kao što možete videti na slici 14.4.

SLIKA 14.4 *Poruka o grešci koja se prikazuje kada pokrenete primer CallCpp, a Windows ne može da pronađe neophodni DLL*

Kreiranje DLL-a u Delphiju

Pored upotrebe DLL-ova napisanih u drugim okruženjima, možete upotrebiti Delphi za izradu DLL-ova koji se mogu koristiti u Delphi programima, ili uz bilo koji alat za programiranje koji podržava DLL-ove. Izrada DLL-ova u Delphiju je toliko laka, da se može desiti da ovu mogućnost previše koristite. Ja Vam predlažem da pokušate da izradite komponente i pakete umesto običnih DLL-ova. Paketi mogu da sadrže komponente, ali takođe i klase, omogućavajući Vam da napišete objektno orijentisani kod koji možete efikasno da koristite više puta. Smeštanje kolekcije funkcija unutar DLL-a je tradicionalniji pristup programiranju, čak i kada funkcije mogu da enkapsuliraju formulare i objekte.

NAPOMENA

Drugim rečima, DLL po definiciji ne podržava objekte u potpunosti, ali Vi to možete da obezbedite, upotrebite Delphi pakete, ili upotrebite Microsoftovu COM tehnologiju (koju ću opisati u narednom poglavlju). ■

Kada pišete DLL, Vi, uopšte uzev, izvozite podrutine, funkcije i procedure. Ukoliko želite da izvezete klase i metode iz Delphi DLL-a, morate da izradite paket (što ću opisati kasnije u ovom poglavlju) ukoliko je potrebno da biblioteku koristite samo iz Delphi programa, ili morate da izradite COM server (ili ActiveX biblioteku), što ćemo videti u narednom poglavlju.

Kada izrađujete komplikovane Delphi aplikacije, koristite objektno orijentisane programske tehnike za definisanje strukture aplikacije. Ukoliko kasnije kod aplikacije podelite na tradicionalne DLL-ove, izgubićete ovu prednost. Korisno je načiniti biblioteke malih funkcija ukoliko se ista funkcija poziva iz različitih okruženja. Uglavnom, možete pisati DLL-ove u kompajliranom jeziku kao što je Object Pascal i pozivati ih iz interpreter okruženja. Naravno, kada god je moguće, najbolje je ceo program izraditi u Delphiju.

Kao što sam već pomenuo, izrada DLL-a je takođe korisna kada je deo koda programa podložan čestim izmenama. U ovom slučaju često možete menjati DLL, a da ostali deo programa ostane nepromenjen. Slično, kada je potrebno da napišete program koji obezbeđuje različite karakteristike za različite grupe korisnika, možete da distribuirate različite verzije DLL-ova korisnicima.

Prvi jednostavni Delphi DLL

Za početnu tačku istraživanja razvoja DLL-ova u Delphiju izabrao sam veoma jednostavnu biblioteku izrađenu u Delphiju. Osnovni cilj ovog primera će biti da se pokaže sintaksa koju treba da koristite za definisanje DLL-a u Delphiju, ali će ilustrovati i nekoliko stvari koje treba uzeti u obzir prilikom prosleđivanja string parametara. Da biste počeli, odaberite komandu File➥New i odaberite DLL opciju na strani New Object Repositoryja. Ovim se kreira veoma jednostavan izvorni fajl koji počinje sledećom definicijom:

```
library Project1;
```

Iskaz `library` označava da želimo da izradimo DLL umesto izvršnog fajla. Sada biblioteci možemo da dodamo rutine i izlistamo ih u iskazu `exports`:

```
function Triple (N: Integer): Integer; stdcall;
begin
  Result := N * 3;
end;

function Double (N: Integer): Integer; stdcall;
begin
  Result := N * 2;
end;

exports
  Triple, Double;
```

U ovoj osnovnoj verziji DLL-a nije potrebno da koristimo iskaz `uses`; ali, u opštem slučaju, fajl glavnog projekta sadrži samo `exports` iskaze, dok se deklaracije funkcija smeštaju u zasebnu jedinicu. U konačnom izvornom kodu primera FirstDLL (verzije koju ste preuzeli), ja sam, zapravo, malo promenio kod u odnosu na verziju koja je ovde prikazana, da bih prikazao poruku svaki put kada se funkcija pozove. Postoje dva načina da se ovo postigne. Najjednostavniji način je da kod promenite na sledeći način:

```
uses
  Dialogs;
function Triple (N: Integer): Integer; stdcall;
begin
  ShowMessage ('Triple function called');
  Result := N * 3;
end;
```

Ovaj kod zahteva da Delphi poveže dosta VCL koda u aplikaciju. Ukoliko statički povežete VCL u ovaj DLL, rezultujuća veličina bi bila oko 200KB. Razlog je što funkcija `ShowMessage` prikazuje VCL formular koji sadrži VCL kontrole i koristi VCL grafičke klase, a one se direktno referišu na VCL sistem usmeravanja i VCL aplikaciju i objekte ekrana. Za ovaj jednostavan slučaj, bolje je poruku prikazati upotrebom direktnih API poziva, tako da VCL kod nije neophodan:

```
uses
  Windows;

function Triple (N: Integer): Integer; stdcall;
begin
  MessageBox (0, 'Triple function called',
```

```
    'First DLL', mb_OK);
  Result := N * 3;
end;
```

Ova promena koda smanjuje veličinu aplikacije za oko 20KB. U preuzetom izvornom kodu primera FirstDLL pronaći ćete obe verzije biblioteke, one koje smo komentarisali. Promenom sekcije komentara lako možete da promenite kod i sami eksperimentišete.

NAPOMENA

Ova velika razlika u veličini ističe činjenicu da ne treba previše da koristite DLL-ove u Delphiju, da izbegavate kompajliranje koda VCL-a u više izvršnih fajlova. Naravno, Vi možete da smanjite veličinu Delphi DLL-a upotrebom paketa za vreme izvršavanja, što ćemo razmatrati kasnije u ovom poglavlju. ■

Ukoliko upotrebom API verzije DLL-a pokrenete test program kao što je primer CallFirst (opisaću ga kasnije), njegovo ponašanje neće biti korektno. Zapravo, možete kliknuti kontrole kojima se DLL funkcije pozivaju više puta, a da prethodno ne zatvorite okvire sa porukom koje DLL prikazuje. Ovo se dešava zato što je prvi parametar `MessageBox` API poziva nula. Njegova vrednost bi, zapravo, trebalo da bude hendl glavnog formulara programa ili formulara aplikacije. Mi ćemo načiniti ovu izmenu, mada iz drugih razloga, u narednom primeru, nazvanom FormDLL.

Više funkcija istog imena u Delphi DLL-ovima

Kada kreirate DLL u jeziku C++, više funkcija istog imena — overloaded functions (i generalno sve funkcije kompajlirane uz pomoć C++ kompajlera) koriste izdvajanje naziva za generisanje različitog naziva za svaku funkciju, uključujući tip parametara u nazivu, kao što smo videli u primeru CppDll.

Kada kreirate DLL u Delphiju i koristite overloaded funkcije, (dakle, više funkcija istog imena označenih direktivom `overload`) Delphi Vam omogućava da izvezete samo jednu od ovih funkcija. Potrebno je da naznačite koju funkciju izvozite, klauzulom `exports`, što je pokazano delom koda FirstDLL:

```
function Triple (C: Char): Integer; stdcall; overload
begin
  ShowMessage ('Triple (Char) function called');
  Result := Ord (C) * 3;
end;

function Triple (N: Integer): Integer; stdcall; overload;
begin
  ShowMessage ('Triple (Integer) function called');
  Result := N * 3;
end;

exports
  Triple (N: Integer);
```

NAPOMENA

Obrnuto je takođe moguće: možete da uvezete niz sličnih funkcija iz DLL-a i sve ih definišete kao overloaded funkcije u Pascal deklaraciji. Delphijeva jedinica OpenGl.PAS sadrži niz primera ove tehnike. ■

Izvoženje stringova iz DLL-a

Uopšte, funkcije u DLL-u mogu da koriste bilo koji tip parametara i vrate vrednost bilo kog tipa. Postoje dva izuzetka od ovog pravila:

- Ukoliko planirate da DLL pozivate iz nekog drugog programskog jezika, verovatno bi trebalo da koristite Windows osnovne tipove podataka umesto specifičnih Delphi tipova. Na primer, za izražavanje vrednosti boja trebalo bi da koristite celobrojne vrednosti ili Windows `ColorRef` tip umesto Delphijevog osnovnog `TColor` tipa, čineći odgovarajuće konverzije (kao u primeru FormDLL koji ćemo opisati u narednom odeljku). Drugi Delphi tipovi koje zbog kompatibilnosti treba da izbegavate, uključuju objekte, koji se u drugim jezicima uopšte ne mogu koristiti, i Pascal stringove, koji se mogu zameniti `PChar` stringovima. Drugim rečima, svako Windows razvojno okruženje mora da podrži osnovne tipove API-ja i, ukoliko se držite ovih tipova, Vaši DLL-ovi će moći da se koriste i u drugim razvojnim okruženjima.
- Čak i kada planirate da DLL-ove koristite samo iz Delphi aplikacija, ne možete da prosledite Delphi stringove preko granica DLL-a, a da ne preduzmete neke mere obezbeđenja. To je posledica Delphijevog manipulisanja stringovima u memoriji — automatsko alociranje, ponovno alociranje i oslobađanje. Rešenje problema je uključivanje sistemske jedinice ShareMem, kako u DLL-u tako i u programu koji ga koristi. Ova jedinica se mora navesti kao prva jedinica svakog projekta.

U primeru FirstDLL ja sam primenio oba pristupa: jedna funkcija prima i daje Pascal stringove, a druga prima kao parametar `PChar` pokazivač, koji zatim sama funkcija popunjava. Prva funkcija je veoma jednostavna:

```
function DoubleString (S: string; Separator: Char): string; stdcall;
begin
  Result := S + Separator + S;
end;
```

Druga funkcija je prilično komplikovana jer `PChar` stringovi ne mogu koristiti operator + i nisu direktno kompatibilni sa karakterima; separator se mora pretvoriti u string pre nego što se doda. Ovde je kompletan kod; koriste se ulazni i izlazni `PChar` baferi, koji su kompatibilni sa bilo kojim Windows razvojnim okruženjem:

```
function DoublePChar (BufferIn, BufferOut: PChar;
  BufferOutLen: Cardinal; Separator: Char): LongBool; stdcall;
var
  SepStr: array [0..1] of Char;
begin
  // if the buffer is large enough
  if BufferOutLen > StrLen (BufferIn) * 2 + 2 then
  begin
    // copy the input buffer in the output buffer
    StrCopy (BufferOut, BufferIn);
    // build the separator string (value plus null terminator)
    SepStr [0] := Separator;
    SepStr [1] := #0;
    // append the separator
```

```
    StrCat (BufferOut, SepStr);
    // append the input buffer once more
    StrCat (BufferOut, BufferIn);
    Result := True;
  end
  else
    // not enough space
    Result := False;
end;
```

Ova druga verzija koda je svakako mnogo komplikovanija, ali prva se može koristiti samo iz Delphija. Prva verzija zahteva da uključimo jedinicu ShareMem u DLL (i u programe koji koriste DLL) i da prosleđujemo fajl BorlandMM.DLL (naziv je skraćenica za Borland Memory Manager) uz naše programe i biblioteku.

Pozivanje Delphi DLL-a

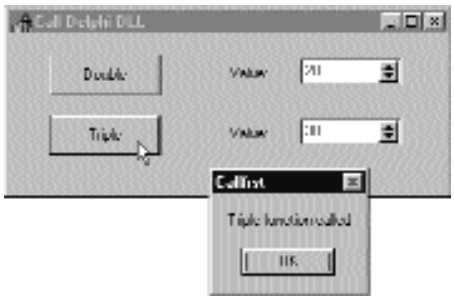
Kako možemo da upotrebimo biblioteku koju smo upravo izradili? Možemo je pozvati iz nekog drugog Delphi projekta ili iz drugih okruženja. Kao primer ja sam izradio projekat CallFirst (koji je smešten u direktorijumu FirstDLL).

Da bismo pristupili DLL funkcijama, moramo ih deklarisati kao `external`, kao što smo radili kada je u pitanju bio C++ DLL. Ovoga puta možemo jednostavno da kopiramo definiciju funkcija iz izvornog koda Delphi DLL-a, dodajući klauzulu `external`:

```
function Double (N: Integer): Integer;
  stdcall; external 'FIRSTDLL.DLL';
```

Ova deklaracija je slična deklaracijama koje smo koristili za poziv C++ DLL-a. Ovoga puta nemamo probleme sa nazivima funkcija. Izvorni kod primera je zaista veoma jednostavan. Kada ponovo deklarišemo funkcije DLL-a kao `external`, možemo ih koristiti kao da su lokalne funkcije. Evo dva primera sa pozivima funkcija koje rade sa stringovima:

```
procedure TForm1.BtnDoubleStringClick (Sender: TObject);
begin
  // call the DLL function directly
  EditDouble.Text :=
    DoubleString (EditSource.Text, ';');
end;

procedure TForm1.BtnDoublePCharClick (Sender: TObject);
var
  Buffer: string;
begin
  // make the buffer large enough
  SetLength (Buffer, 1000);
  // call the DLL function
  if DoublePChar (PChar (EditSource.Text), PChar (Buffer), 1000, '/')
then
    EditDouble.Text := Buffer;
end;
```



SLIKA 14.5 *Izlaz primera CallFirst koji poziva DLL koji smo izradili u Delphiju*

Delphi formular u DLL-u

Pored pisanja jednostavnih DLL-ova sa funkcijama i procedurama, u DLL možete smestiti kompletan formular koji je napisan u Delphiju. To može biti okvir za dijalog ili bilo koji drugi tip formulara, i mogu ga koristiti ne samo Delphi programi, već i druga razvojna okruženja i makro jezici.

Da bih izradio primer FormDLL, izradio sam jednostavan formular sa tri klizača koje možete koristiti za izbor boja i dve oblasti za pregled za rezultujuće boje olovke i četkice. Formular, takođe, sadrži dve bitmapirane kontrole i za svojstvo `BorderStyle` je određena vrednost `bsDialog`. Pored uobičajene izrade formulara, dodao sam i dve nove podrutine jedinici koja definiše formular. U odeljku `interface` ove jedinice dodao sam sledeće deklaracije:

```
function GetColor (Col: LongInt): LongInt; stdcall;
procedure ShowColor (Col: LongInt;
   FormHandle: THandle; MsgBack: Integer); stdcall;
```

U obe podrutine parametar `Col` je početna boja. Primetićete da sam je prosledio kao long integer što odgovara Windowsovom tipu podataka `COLORREF`. Kao što sam ranije pomenuo, upotreba Delphi tipa `TColor` može da prouzrokuje probleme kod aplikacija koje nisu izrađene u Delphiju: iako je `TColor` veoma sličan tipu `COLORREF`, ovi tipovu se ne poklapaju uvek. Kada pišete DLL, ja Vam sugerišem da koristite samo Windowsove osnovne tipove podataka (izuzev ukoliko niste sigurni da će samo Delphi programi koristiti DLL).

Funkcija `GetColor` daje konačnu boju (što je ista boja kao i početna, ukoliko korisnik klikne kontrolu Cancel). Vrednost se momentalno dobija jer funkcija prikazuje formular kao prioritetni formular. Procedura `ShowColor`, umesto toga, prikazuje formular (kao neprioritetni formular). Zbog toga je formularu potreban način da komunicira sa formularom koji ga je pozvao. U ovom slučaju, ja sam odlučio da kao parametar prosledim hendl prozora formulara koji vrši poziv i da ID poruke koristim za povratnu komunikaciju.

U narednim odeljcima ćete videti kako da napišete kod dve podrutine; takođe ćete videti kakvi problemi mogu da nastanu, naročito kada u DLL smestite neprioritetni formular. Naravno, ja ću Vam pokazati i nekoliko alternativa.

Upotreba DLL formulara kao prioritetnog formulara

Kada želite da smestite Delphi komponentu (npr. formular) u DLL, možete da obezbedite samo funkcije kojima se kreira, inicijalizuje ili pokreće komponenta, ili se pristupa svojstvima i podacima komponente. Najjednostavniji pristup je da imate jednu funkciju koja određuje podatke, pokreće komponentu i daje rezultat, kao kod prioritetne verzije. Evo koda funkcjie koja je dodata odeljku `implementation` jedinice koja definiše formular:

```
function GetColor (Col: LongInt): LongInt; stdcall;
var
  FormScroll: TFormScroll;
begin
  // default value
  Result := Col;
  try
    FormScroll := TFormScroll.Create (Application);
    try
      // initialize the data
      FormScroll.SelectedColor := Col;
      // show the form
      if FormScroll.ShowModal = mrOK then
        Result := FormScroll.SelectedColor;
    finally
      FormScroll.Free;
    end;
  except
    on E: Exception do
      MessageDlg ('Error in FormDLL: ' +
        E.Message, mtError, [mbOK], 0);
  end;
end;
```

Važan element je struktura funkcije `GetColor`. Kod kreira formular na početku, određuje neke početne vrednosti, a zatim pokreće formular i na kraju izdvaja konačne podatke. Ono što ovaj kod čini drugačijim od koda koji uobičajeno pišemo jeste upotreba obrade izuzetaka.

- Blok `try-except` štiti celu funkciju, tako da će biti uočen bilo koji izuzetak koji funkcija generiše i prikazaće se odgovarajuća poruka. Razlog za obradu bilo kojeg mogućeg izuzetka je taj da aplikacija koja poziva može biti napisana u bilo kojem jeziku, možda u jeziku koji ne može da obradi izuzetke. Čak i kada poziv dolazi iz Delphi programa, ponekad je korisno da se upotrebi zaštitni pristup.
- Blok `try-finally` štiti operacije na formularu osiguravajući da će se objekat formulara pravilno ukloniti, čak i kada se pozove izuzetak. Ovakav tip koda se često koristi u Delphi programima kao i u DLL-ovima.

Proverom vrednosti koju daje funkcija `ShowModal`, program određuje rezultat funkcije. Ja sam unapred odredio vrednost pre ulaska u blok try da bih obezbedio da se uvek izvrši (i da bih izbegao upozorenje kompajlera da je rezultat funkcije možda nedefinisan).

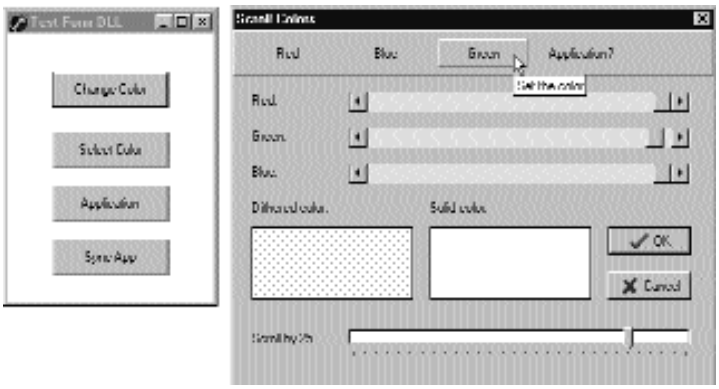
Sada kada smo ažurirali formular i napisali kod jedinice, možemo preći na izvorni kod projekta, koji (privremeno) postaje:

```
library FormDLL;

uses
  ScrollF in 'SCROLLF.PAS' {FormScroll};

exports
  GetColor;
end.
```

Sada možemo da upotrebimo Delphi program za testiranje formulara koji smo smestili u DLL. Primer UseCol se nalazi u istom direktorijumu kao i prethodni DLL, FormDLL (i oba projekta i deo grupe FormDLL, fajl FormDll.BPG). Formular primera UseCol sadrži kontrolu za poziv funkcije `GetColor` koja se nalazi u DLL-u. Evo definicije te funkcije i koda metoda `Button1Click`:

```
function GetColor (Col: LongInt): LongInt;
  stdcall; external 'FormDLL.DLL';

procedure TForm1.Button1Click (Sender: TObject);
var
  Col: LongInt;
begin
  Col := ColorToRGB (Color);
  Color := GetColor (Col);
end;
```

Pokretanjem ovog programa (videti sliku 14.6) prikazuje se okvir za dijalog, koji koristi trenutnu boju pozadine glavnog formulara. Ukoliko promenite boju i kliknete OK, program koristi novu boju kao boju pozadine glavnog formulara.



SLIKA 14.6 *Izvršavanje test programa UseCol kada poziva okvir za dijalog koji smo smestili u FormDLL*

Ukoliko ovo izvršite kao neprioritetni okvir za dijalog, gotovo sve karakteristike formulara funkcionišu dobro. Možete videti oblačiće, ravne kontrole na paleti alata se ponašaju pravilno i ne dobijate nikakav dodatni element na liniji aktivnih procesa. Ovo je možda očigledno, ali se neće dogoditi kada formular unutar DLL-a koristimo kao neprioritetni. Čak i sa prioritetnim

formularima ja preporučujem sinhronizaciju objekata aplikacije iz DLL-a i izvršnog fajla, kao što je opisano u narednom odeljku.

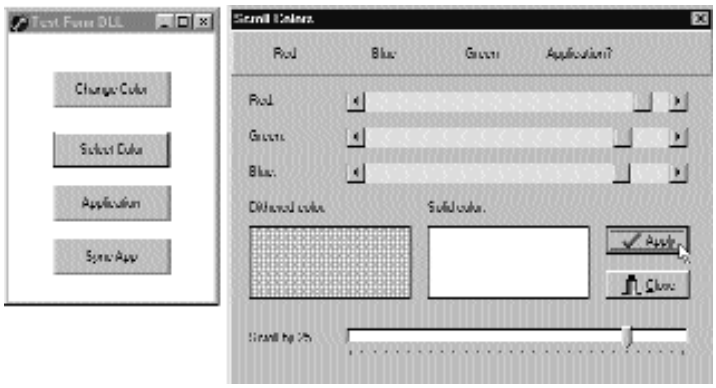
Neprioritetni formular u DLL-u

Druga podrutina primera FormDLL koristi drugačiji pristup. Kao što je istaknuto, podrutina dobija tri parametra: boju, hendl glavnog formulara i broj poruke za obaveštavanje kada se promeni boja. Ove vrednosti se čuvaju u privatnim podacima formulara:

```
procudure ShowColor (Cal: LongInt;
  FormHandle: THandle; MsgBack: Integer); stdcall;
var
  FormScroll: TFormScroll;
begin
  FormScroll := TFormScroll Create (Application);
  try
    // initialize the data
    FormScroll.FormHandle = FormHandle;
    FormScroll.MsgBack := MsgBack;
    FormScroll.SelectedColor := Col;
    // show the form
    FormScroll Show;
  except
    on E: Exception do
    begin
      MessageDlg ('Error in EormDLL: ' +
        E.Message, mtError, [mbOK], 0);
      FormScroll Free;
    end;
  end;
end;
```

Kada je formular aktiviran, on proverava da li je kreiran kao prioritetni (testiranjem polja `FormHandle`). U ovom slučaju formular menja zaglavlje i ponašanje kontrole OK, kao i stil kontrole Cancel (modifikovane kontrole možete videti na slici 14.7):

```
procedure TFormScroll.FormActivate(Sender: TObject);
begin
  // change buttons for modeless form
  if FormHandle <> 0 then
  begin
    BitBtn1.Caption := 'Apply'
    BitBtnl.OnClick := ApplyClick;
    BitBtn2.Kind := bkClose;
  end;
end;
```



SLIKA 14.7 *Kada se DLL formular koristi kao prioritetni, njegove kontrole su malo izmenjene (kao što možete uočiti kada ovu sliku uporedite sa slikom 14.6)*

Metod `ApplyClick`, koji sam ručno dodao formularu, šalje poruku obaveštenja glavnom formularu i koristi jedan od parametara da pošalje odabranu boju:

```
procedure TFormScroll.ApplyClick(Sender: TObject);
begin
  // notify to the main form
  SendMessage (FormHandle, MsgBack, SelectedColor, 0);
end;
```

Konačno, događaj formulara OnClose uklanja objekat formulara:

```
procedure TFormScroll.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
  // used by the modeless form
  Action := caFree;
end;
```

Vratimo se sada na demo program. Druga kontrola primera UseForm ima sledeći kod:

```
procedure TForml.BtnSelectClick(Sender: Tobject);
var
  Col: Longlnt;
begin
  Col := ColorToRCB (Color);
  ShowColor (Col, Handle, wm_user);
end;
```

Formular, takođe, sadrži kod za obradu poruka koji je povezan sa porukom `wm_user`. Ovaj metod čita vrednost parametra koji odgovara boji i određuje ga:

```
procedure TForm1.UserMessage(var Msg: Message);
begin
  Color := Msg.WParam;
end;
```

Izvršavanje ovog programa proizvodi čudne efekte. U osnovi, prioritetni formular i glavni formular nisu sinhronizovani, te se oba prikazuju na Windows TaskBaru; i kada minimizirate glavni formular, drugi formular ostaje na ekranu. Dva formulara se ponašaju kao da su deo zasebnih aplikacija, a razlog je što dva Delphi programa (DLL i EXE) imaju dva zasebna globalna objekta `Application`, a samo objekat `Application` izvršnog fajla je povezan sa prozorom.

Da bih testirao ovu situaciju, dodao sam kontrolu glavnom formularu i formularu iz DLL-a, prikazujući numeričku vrednost hendla objekta `Application`. Evo koda jedne od kontrola:

```
procedure TFormScroll.spApplicationClick(Sender: TObject);
begin
  ShowMessage ('Application Handle: ' +
    IntToStr (Application.Handle));
end;
```

Za formular u DLL-u neizostavno ćete dobiti vrednost 0, dok ćete za formular u izvršnom fajlu dobiti numeričku vrednost koju svaki put određuje Windows.

Da bismo popravili problem, DLL-u možemo da dodamo inicijalizacionu funkciju koja prosleđuje hendl prozora aplikacije biblioteci. Praktično kopiramo `Handle` objekta `Application` izvršnog fajla u isto svojstvo objekta `Application` DLL-a. Ovo je dovoljno da se obavi sinhronizacija dva `Application` objekta i da učinimo da se formulari ponašaju kao jednostavan Delphi program. Evo koda funkcije iz DLL-a:

```
procedure SyncApp (AppHandle: THandle); stdcall;
begin
  Application.Handle := AppHandle;
end;
```

a evo i poziva na njega iz izvršnog fajla:

```
procedure TForm1.BtnSyncClick(Sender: TObject);
begin
  SyncApp (Application.Handle);
  BtnSync.Enabled := False;
end;
```

NAPOMENA

Dodeljivanje hendla objekta aplikacije iz DLL-a nije zaobilaženje problema, već dokumentovana operacija koju zahteva VCL. VCL `Application` objekat podržava dodeljivanje svojstva `Handle` (za razliku od većine Handle svojstava VCL-a) naročito da bi se omogućilo programerima da DLL formulare vežu za okruženje aplikacije koja ih koristi. ■

Ja sam ovaj kod povezao sa kontrolom, umesto da ga automatski izvršim prilikom pokretanja, da bih korisnicima omogućio da testiraju ponašanje u dva različita slučaja. Pre nego što kliknete kontrolu *Sync App*, sekundarni neprioritetni formular se čudno ponaša. Ukoliko ga zatvorite, sinhronizujete aplikacije, a kada zatim kreirate drugu instancu neprioritetnog formulara, ponašaće se gotovo korektno. Jedini problem koji se vidi je da ravne kontrole neprioritetnog formulara neće biti istaknute kada se pokazivač miša nađe iznad njih. Videćemo kako da rešimo ovaj problem upotrebom paketa samo za vreme izvršavanja na kraju ovog poglavlja.

NAPOMENA

Tehnički, ovo ponašanje točkića zavisi od činjenice da kontrole u DLL-u neće prihvatiti poruke `cm_MouseEnter` i `cm_MouseLeave` jer se metod `Application.Idle` DLL-a nikada ne poziva. Objekat `Application` DLL-a, zapravo, ne izvršava petlju poruka aplikacije. Možete da ga aktivirate izvođenjem funkcije koja poziva internu rutinu `Application.Idle` iz DLL-a, i da tu funkciju pozovete iz aplikacije domaćina kada petlja poruka nema posla. Kao što sam istakao, svi ovi problemi (i nekoliko drugih) mogu se rešiti upotrebom paketa samo za vreme izvršavanja. ■

Pozivanje Delphi DLL-a iz Visual Basica za aplikacije (VBA)

Ovaj okvir za dijalog za boje možemo prikazati i iz drugih programskih jezika. Pozivanje ovog DLL-a iz jezika C ili C++ je lako. Da biste povezali aplikaciju, potrebno je da generišete uvoznu biblioteku (upotrebom pomoćnog programa `IMPLIB`) i da dodate rezultujući LIB fajl projektu. Kako sam ja već koristio C++ kompajler u ovom poglavlju, ovoga puta ću napisati sličan primer upotrebom Microsoft Worda za Windows i Visual Basica za aplikacije (VBA).

Da biste počeli, otvorite Microsoft Word. Zatim otvorite Wordov Macro okvir za dijalog (upotrebom elementa menija Tools➥Macro ili sličnom komandom, u zavisnosti od Vaše verzije Worda), unesite novi naziv makroa, npr. **DelphiColor**, i kliknite Create. Sada možete da napišete BASIC kod, koji deklariše funkciju našeg DLL-a i koji je poziva. BASIC makro koristi rezultat DLL funkcije na dva načina. Pozivom `Insert` makro dodaje aktuelnom dokumentu opis boje koristeći količinu crvene (Red), zelene (Green) i plave (Blue) boje, a pozivom `Print` prikazuje numeričku vrednost na statusnoj liniji:

```
Declare Function GetColor Lib "FormDLL" (Col As Long) As Long
Sub MAIN
  NewColor = GetColor(0)
  Print "The code of the color is " + Str$(NewColor)
  Insert "Red: " + Str$(NewString Mod 256) + Chr$(13)
  Insert "Green: " + Str$(Int(NewString / 256) Mod 256) + Chr$(13)
  Insert "Blue: " + Str$(Int(NewString / (256 * 256))) + Chr$(13)
End Sub
```

Na nesreću, ne postoji lak način za upotrebu RGB boja u Wordu jer su njegove šeme boja zasnovane na fiksiranim kodovima boja. Evo primera izlaza ovog makroa:

```
Red: 141
Green: 109
Blue: 179
```

Dobra stvar je da sam proizveo tri linije teksta, koje su prikazane iznad izvršavanjem makroa, dok sam pisao tekst ovog poglavlja. Tekst ovog makroa možete pronaći u fajlu WORDCALL.TXT, u poddirektorijumu koji sadrži ovaj DLL. Ukoliko želite da ga testirate, ne zaboravite da prvo kopirate DLL fajl u jedan od direktorijuma putanje ili u Windowsov sistemski direktorijum. Naravno, potreban Vam je Microsoft Word da biste izvršili ovaj program. Ipak, ostale aplikacije Microsoft Officea (i makro jezici ostalih Office aplikacija) će verovatno zahtevati sličan kod.

NAPOMENA

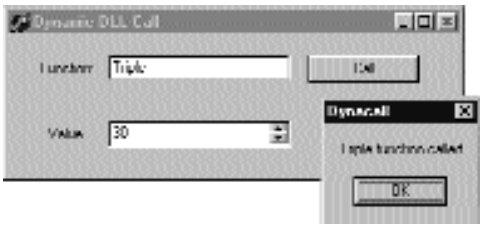
Bolji način integracije Delphi koda sa Office aplikacijama je upotreba OLE Automationa, umesto pisanja DLL-ova i njihovog pozivanja iz makro jezika. Primere OLE Automationa ćemo videti u Poglavlju 16. ■

Pozivanje DLL funkcije u vreme izvršavanja

Sada kada znamo kako da pristupimo resursima DLL-a u vreme izvršavanja, možemo da upotrebimo ovaj pristup za pristup funkciji. Ja sam izradio veoma jednostavan primer koji ovo pokazuje i načinio sam ga veoma fleksibilnim. Prvo ćemo pogledati primer, a zatim ćemo razmotriti opšte slučajeve ovog pristupa koji mogu biti korisni. Primer je nazvan DynaCall i koristi biblioteku FirstDLL koja je izrađena ranije u ovom poglavlju (da bi program funkcionisao ja sam kopirao DLL u isti direktorijum kao i primer DynaCall). Umesto deklarisanja funkcija `Double` i `Triple` i njihove direktne upotrebe, ovaj primer postiže isti efekat nešto složenijim kodom. Prednost je u tome što, ukoliko se nove funkcije dodaju DLL-u, nećemo ponovo morati da pregledamo izvorni kod programa i ponovo ga kompajliramo da bismo pristupili novim funkcijama.

Formular koji je prikazan u ovom primeru sadrži kontrolu, polje za izmene i komponentu SpinEdit. Kada kliknete kontrolu, izvršava se jedini metod programa. Prvo, metod poziva funkciju `LoadLibrary`. Zatim, ukoliko je hendl instance biblioteke korektan, program poziva API funkciju `GetProcAddress`. Ova funkcija pretražuje exports tabelu DLL-a u potrazi za nazivom funkcije koji je prosleđen kao parametar. Ukoliko funkcija `GetProcAddress` pronađe naziv, daje pokazivač na zahtevanu proceduru. Sada jednostavno možemo da konvertujemo pokazivač funkcije u odgovarajući tip podataka i pozovemo je. Izlaz programa i efekat ovog poziva su prikazani na slici 14.8. Evo (prilično komplikovanog) koda:

```
type
TIntFunction = function (I: Integer): Integer; stdcall;

const
  DllName = 'Firstdll.dll';

procedure TForml.Button1Click(Sender: TObject);
var
  HInst: THanle;
  FPointer: TFarProc;
  MyFunct: TIntFunction;
begin
  HInst := LoadLibrary (DllName);
  if HInst > 0 then
  try
    FPointer := GetProcAddress (HInst,
      PChar (Edit1.Text));
    if FPointer <> nil then
    begin
      MyFunct := TIntFunction (FPointer);
      SpinEdit1.Value := MyFunct (SpinEdit1.Value);
    end
    else
      ShowMessage (Edit1.Text + ' DLL function not found';);
  finally
    FreeLibrary (HInst);
  end
  else
    ShowMessage (DllName + ' library not found');
end;
```



SLIKA 14.8 *Izlaz programa DynaCall*

Kako pozvati proceduru u Delphiju kada već imate pokazivač na proceduru? Jedna mogućnost je da pokazivač konvertujete u proceduralni tip i da zatim pozovete proceduru upotrebom promenljive proceduralnog tipa, kao u prethodnom listingu. Primetićete da proceduralni tip koji definišete mora biti kompatibilan sa definicijom procedure u DLL-u. Ovo je Ahilova peta ovog metoda — nema provere tipova parametara.

Koje su prednosti ovog pristupa? U teoriji, možete ga koristiti za pristupanje bilo kojoj funkciji bilo kog DLL-a, bilo kada. U praksi, koristan je kada imate različite DLL-ove sa kompatibilnim funkcijama ili jedan DLL sa nekoliko kompatibilnih funkcija, kao što je to kod nas slučaj. Ono što možemo učiniti je da pozovemo metode `Double` i `Triple` unošenjem njihovih naziva u polje za izmene. Sada, ukoliko nam neko da DLL sa novim funkcijama koje kao parametar primaju celobrojnu vrednost i kao rezultat daju celobrojnu vrednost, možemo ih pozvati unošenjem naziva funkcija u polje za izmene. Ne moramo čak ni ponovo da kompajliramo aplikaciju.

Ovim kodom kompajler i linker ignorišu postojanje DLL-a. Kada se program učita, DLL se ne učitava odmah. Mi program možemo učiniti još fleksibilnijim i omogućiti korisniku da unese naziv DLL-a koji želi da koristi. U nekim slučajevima ovo je velika prednost. Program može da menja DLL-ove u vreme izvršavanja; to je nešto što direktan pristup ne dozvoljava. Primetićete da je ovaj pristup učitavanja DLL funkcija uobičajen u makro jezicima i da se koristi u mnogim vizuelnim programskim okruženjima. Takođe, kod Wordovog makroa koji smo videli ranije u ovom poglavlju, koristi ovaj pristup za učitavanje DLL-a i za pozivanje spoljašnjih funkcija. Dakle, Vi ne želite da ponovo kompajlirate Word, zar ne?

Samo sistem baziran na kompajleru i linkeru, kakav je Delphi, može da koristi direktan pristup, koji je, uopšte uzev, pouzdaniji i nešto brži. Ja smatram da je indirektni pristup primera DynaCall koristan samo u specijalnim slučajevima, ali može biti izuzetno moćan.

DLL u memoriji: kod i podaci

Mi možemo da koristimo ovu tehniku, zasnovanu na API funkciji `GetProcAddress`, za pronalaženje memorijske adrese na kojoj je mapirana funkcija aktuelnog procesa, upotrebom sledećeg koda:

```
procedure TForm1.Button3Click (Sender: TObject);
var
  HDLLInst: THandle;
begin
  HDLLInst := LoadLibrary ('dllmem');
```

```
  Label1.Caption := Format ('Address: %p', [
    GetProcAddress (HDLLInst, 'SetData')]);
  FreeLibrary (HDLLInst);
end;
```

Ovaj kod u oznaci (labeli) prikazuje memorijsku adresu funkcije, koja se nalazi unutar adresnog prostora aplikacije koja je poziva. Ukoliko pokrenete dva programa koja koriste ovaj kod, oni će u opštem slučaju prikazati istu adresu. Ovim se pokazuje da je kod u memoriju učitan samo jednom na jednoj memorijskoj adresi. Čak i kada je kod učitan samo jednom, memorijska adresa će biti drugačija ukoliko DLL treba da se premesti u jedan od procesa, ali ne i u drugi, ili ukoliko su oba procesa premestila DLL na neku drugu osnovnu adresu.

Ukoliko je kod DLL učitan samo jednom, šta se dešava sa globalnim podacima? U osnovi, svaka kopija DLL-a sadrži sopstvenu kopiju podataka u adresnom prostoru aplikacije koja poziva DLL. Ipak, zaista je moguće deliti globalne podatke između aplikacija upotrebom DLL-ova. Najčešća tehnika za deljenje podataka je upotreba fajlova koji su mapirani u memoriji. Ja ću koristiti ovu tehniku za DLL, ali tehnika može da se koristi za deljenje podataka između aplikacija.

Ovaj primer je nazvan DllMem i koristi projekat grupu istog naziva, kao što je bio slučaj i sa prethodnim primerima ovog poglavlja. Projekat grupa DllMem sadrži projekat DllMem (sam DLL) i projekat UseMem (demo aplikaciju).

DLL kod sadrži jednostavan fajl projekta, kojim se izvoze četiri podrutine:

```
library dllmem

uses
  SysUtils,
  DllMemU in 'DllMemU.pas';

exports
  SetData, GetData,
  GetSharedData, SetSharedData;
end.
```

Kod se nalazi u sekundarnoj jedinici (DllMem.PAS) koja sadrži kod četiri podrutine koje čitaju i zapisuju dve globalne memorijske lokacije. Ove lokacije čuvaju celobrojnu vrednost i pokazivač na celobrojnu vrednost. Evo deklaracija promenljive i dve Set rutine:

```
var
  PlainData: Integer = 0; // not shared
  ShareData: ^Integer; // shared

procedure SetData (I: Integer); stdcall;
begin
  PlainData := I;
end;

procedure SetSharedData (I: Integer); stdcall;
begin
  Sharedata^ := I;
end;
```

Deljenje podataka upotrebom fajlova koji su mapirani u memoriji

Za podatke koji nisu deljeni nema šta više da se učini. Da bi se pristupilo deljenim podacima, s druge strane, DLL treba da kreira fajl koji je mapiran u memoriji i da zatim dobije pokazivač na tu memorijsku oblast. Ove dve operacije zahtevaju dva Windows API poziva:

- `CreateFileMapping` zahteva kao parametre naziv fajla (ili `$FFFFFFFF` za upotrebu virtuelnog fajla u memoriji), neke atribute zaštite i sigurnosti, veličinu podataka i interni naziv (koji mora biti isti da bi se mapirani fajl delio između više aplikacija koje vrše pozive).
- `MapViewOfFile` zahteva kao parametre hendl fajla koji je mapiran u memoriji, neke atribute i offsete i veličinu podataka (još jednom).

Evo izvornog koda odeljka `initialization`, koji se izvršava svaki put kada se DLL učita u prostor novog procesa (to jest, po jednom za svaku aplikaciju koja koristi DLL):

```
var
  hMapFile: THandle;

const
  VirtualFileName = 'ShareDllData';
  DataSize = sizeof (Integer);

initialization
  // create memory mapped file
  hMapFile := CreateFileMapping (SFFFFFFFF, nil,
    Page ReadWrite, 0, DataSize, VirtualFileName);
  if hMapFile = 0 then
    raise Exception.Create ('Error creating memory mapped file');

  // get the pointer to the actual data
  ShareData := MapViewOfFile (
    hMapFile, File_Map_Write, 0, 0, DataSize);
```

Kada se završi izvršavanje aplikacije i kada se oslobodi DLL, kod mora da oslobodi pokazivač na mapirani fajl i na samo mapiranje fajla:

```
finalization
  UnmapViewOfFi1e (ShareData);
  CloseHandle (hMapFile);
```

Kod programa koji koristi ovaj DLL, program UseMem, veoma je jednostavan. Formular ove aplikacije sadrži četiri polja za izmene (od kojih su dva povezana sa kontrolom UpDown), pet kontrola i oznaku. Prva kontrola čuva vrednost prvog polja za izmene među DLL podacima, uzimajući vrednost povezane kontrole UpDown:

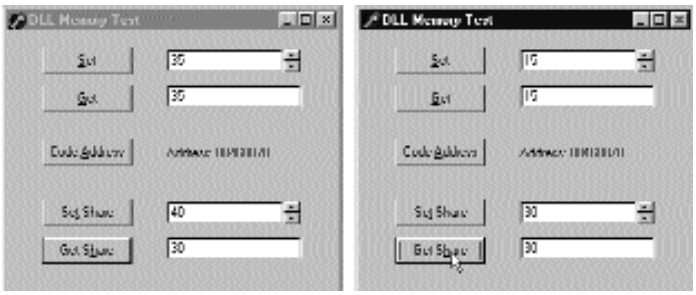
```
procedure TForm1.Button1Click(Sender TObject);
begin
  SetData (UpDown1.Position);
end;
```

Ukoliko kliknete drugu kontrolu, program kopira DLL podatke u drugo polje za izmene:

```
procedure TForm1.Button2Click(Sender TObject);
begin
  Edit2.Text := IntToStr(GetData);
end;
```

Treća kontrola se koristi za prikazivanje memorijske adrese funkcije, čiji kod je prikazan na početku ovog odeljka, a poslednje dve kontrole imaju u osnovi isti kod kao i prve dve, ali pozivaju proceduru `SetSharedData` i funkciju `GetSharedData`.

Ukoliko pokrenete dve kopije ovog programa, možete videti da svaka kopija sadrži sopstvenu vrednost za čiste globalne podatke DLL-a, dok je vrednost deljenih podataka zajednička. Odredite različite vrednosti u programima, a zatim ih preuzmite i videćete šta hoću da kažem. Ova situacija je ilustrovana slikom 14.9.



SLIKA 14.9 *Ukoliko pokrenete dve kopije programa UseMem, videćete da globalni podaci u DLL-u nisu deljeni*

UPOZORENJE

Fajlovi koji su mapirani u memoriji rezervišu najmanje 64KB opsega virtuelnih adresa i zauzimaju fizičku memoriju u 4KB strana. Primer upotrebe 4-bajtnog celobrojnog podatka u deljenoj memoriji je prilično skup, naročito ukoliko koristite isti pristup za deljenje više vrednosti. Ukoliko je potrebno da delite nekoliko promenljivih, trebalo bi sve da ih smestite u jednu deljenu memorijsku oblast (pristupajući različitim promenljivima upotrebom pokazivača ili izradom strukture sloga za sve promenljive). ■

Upotreba Delphi paketa

U Delphiju su paketi komponenata važan tip DLL-ova. Paketi Vam omogućavaju da upakujete grupu komponenata i da zatim komponente povežete statički (dodavanjem njihovog kompajliranog koda izvršnom fajlu Vaše aplikacije) ili dinamički (zadržavajući kod komponente u DLL-u, paketu samo za vreme izvršavanja koji distribuirate uz svoj program). U prethodnom poglavlju smo videli kako da izradimo paket. Sada želim da istaknem neke prednosti i nedostatke dva načina povezivanja paketa. Postoje mnogi elementi koje treba da imate na umu:

- Upotreba paketa kao DLL-ova čini izvršni fajl daleko manjim.
- Upotreba statički povezanih paketa Vam omogućava da distribuirate samo deo koda paketa. Uopšte uzev, veličina izvršnog fajla aplikacije i veličina potrebnog

paketa DLL-a koji se zahteva mnogo je veća od veličine statički povezanog programa. Linker uključuje samo kod koji se zaista koristi u programu, dok paket mora da poveže sve funkcije i klase deklarisane u interfejs odeljcima svih jedinica koje se nalaze u paketu.

- Ukoliko distribuirate nekoliko Delphi aplikacija zasnovanih na istom paketu, možda ćete distribuirati manje koda jer se paketi samo za vreme izvršavanja dele. Drugim rečima, kada korisnici Vaših aplikacija imaju standardne Delphi pakete za vreme izvršavanja, možete im davati veoma male programe. Ovo Vam čak omogućava da programe distribuirate preko Interneta.
- Ukoliko pokrenete nekoliko Delphi aplikacija koje se zasnivaju na istim paketima, možete da uštedite memorijski prostor u vreme izvršavanja; kod paketa samo za vreme izvršavanja u memoriju se učitava jednom između više Delphi aplikacija.
- Ne brinite previše zbog distribucije velikih izvršnih fajlova. Imajte na umu da kada načinite male izmene u programu, možete da upotrebite bilo koji od virtuelnih alata za kreiranje fajla "zakrpe" (patch file), te možete da distribuirate samo fajl koji sadrži razlike, ne celokupnu kopiju fajlova.
- Ukoliko smestite nekoliko formulara aplikacije u pakete samo za vreme izvršavanja, možete da ih delite u više aplikacija. Kada izmenite ove formulare, u opštem slučaju je potrebno da ponovo kompajlirate i glavni program i da ponovo distribuirate korisnicima i program i pakete. Naredni odeljak detaljno opisuje ovu složenu temu.

Prevođenje paketa (verzije paketa)

Veoma važan i često pogrešno shvaćen element je distribucija ažuriranih paketa. Kada ažurirate DLL, možete poslati novu verziju, a izvršni programi koji se pozivaju na DLL će u opštem slučaju funkcionisati (izuzev ukoliko niste uklonili postojeće izvezene funkcije, ili ukoliko niste promenili neke od prametara postojećih funkcija).

Kada distribuirate Delphi paket, ukoliko paket ažurirate i bilo šta promenite u interfejs odeljku bilo koje izvezene jedinice paketa, potrebno je da ponovo kompajlirate sve aplikacije koje koriste taj paket. Možete samo da promenite kod u implementacionom odeljku jedinica paketa ukoliko želite da izbegnete ponovno kompajliranje izvršnih fajlova koji koriste paket.

DCU fajlovi u Delphiju sadrže tag verzije koji se zasniva na sumi koja se izračunava iz interfejs odeljka jedinice. Kada promenite interfejs odeljak jedinice, svaka druga jedinica koja se bazira na toj jedinici mora se ponovo kompajlirati. Kompajler poredi sumu jedinice prethodnih kompajliranja sa novom sumom i odlučuje da li se zavisna jedinica mora ponovo kompajlirati. To je razlog zbog kojeg morate ponovo da kompajlirate svaku jedinicu kada dobijete novu verziju Delphija, koja je izmenila sistemske jedinice.

Paketi su kolekcija sistemskih jedinica. U Delphiju 3 suma paketa, koja se dobija iz suma jedinica koje paket sadrži i suma paketa koji su potrebni, dodavala se kao dodatna ulazna funkcija paketu biblioteke, tako da svaki izvršni fajl zasnovan na starijoj verziji paketa ne bi mogao da se pokrene.

Delphi 4 i Delphi 5 sadrže komfornije veze paketa u vreme izvršavanja. Veze DCU fajlova u vreme dizajniranja ostaju identične. Suma za proveru paketa se više ne proverava, pa jedinice koje su deo paketa možete direktno menjati i možete prosleđivati novu verziju paketa za upotrebu sa postojećim izvršnim fajlovima. Kako se metodima pristupa po nazivu, ne možete ukloniti ni jedan od postojećih metoda. Ne možete promeniti čak ni njegove parametre zbog tehnika izdvajanja naziva.

Uklanjanje metoda na koji se referiše iz programa koji vrši poziv, prekinuće program tokom procesa učitavanja. Ukoliko načinite druge izmene, može se desiti da program neočekivano prekine izvršavanje. Na primer, ukoliko neku komponentu zamenite sličnom komponentom, program koji vrši poziv će možda još uvek moći da pristupi komponenti u tom memorijskom prostoru, iako je to sada drugačija komponenta.

Ukoliko se odlučite na ovaj nepouzdani put promene interfejs jedinica paketa, a da ponovo ne kompajlirate sve programe koji koriste paket, najmanje što možete učiniti je da ograničite izmene. Kada dodajete nova svojstva ili nevirtuelne metode formularu, trebalo bi da imate mogućnost da u potpunosti održite kompatibilnost sa postojećim programima koji već koriste paket. Takođe, dodavanje polja i virtuelnih metoda može uticati na internu strukturu klase, što dovodi do problema sa postojećim programima koji očekuju drugačiju klasu i VMT izlaz. Naravno, ovo se odnosi na binarnu kompatibilnost između EXE i BPL. Bilo kakva izmena interfejsa jedinice paketa raskida DCU/DCP kompatibilnost bilo koje jedinice koja se referiše na Vaš paket.

UPOZORENJE

Ovde govorim o distribuciji kompajliranih programa koji su podeljeni na EXE i pakete, ne na distribuciju komponenata drugim Delphi programerima. Kada je u pitanju distribucija komponenata, pravila prevođenja su striktnija i morate preduzeti dodatne korake u prevođenju paketa. ■

Ja Vam preporučujem da nikada ne menjate interfejs bilo koje jedinice koju izvoze Vaši paketi. Da biste ovo postigli, Vašim paketima možete da dodate jedinicu sa funkcijama za kreiranje formulara (kao što je bio slučaj sa DLL-ovima sa formularima koji ste ranije videli) i da tu jedinicu koristite za pristupanje drugoj jedinici, jedinici koja definiše formular. Mada ne postoji način da se sakrije jedinica koja je povezana u paket, ukoliko nikada direktno ne koristite klasu definisanu u jedinici, već samo ostale rutine, imaćete više fleksibilnosti prilikom izmena. Takođe, možete iskoristiti nasleđivanje za izmenu formulara unutar paketa, a da zapravo ne utičete na originalnu verziju.

Najdrastičnije pravilo za pakete je sledeće pravilo koje koriste programeri komponenata: za dugoročno prosleđivanje i održavanje koda paketa planirajte velike izmene uz manje izmene održavanja. Velike izmene Vaših paketa će zahtevati da svi klijent programi budu ponovo kompajlirani iz izvornog koda; naziv fajla paketa bi trebalo promeniti uz novi broj verzije, a interfejs odeljci jedinica se mogu izmeniti. Verzije radi održavanja paketa treba ograničiti na izmene implementacije, da bi se očuvala potpuna kompatibilnost sa postojećim izvršnim fajlovima i jedinicama.

Izvršni fajlovi i DLL-ovi koji dele VCL pakete

U primeru FormDLL suočili smo se sa problemom: kada formulare smestite u DLL, nećete dobiti očekivano ponašanje za ravne (flat) kontrole, čak i kada sinhronizujete dva objekta aplikacija. Takođe, i izvršni fajl i DLL sadrže kompajlirani kod VCL biblioteke, što dovodi do nepotrebnog dupliranja. Kao što smo ranije razmatrali, najjednostavnije rešenje je da formular kompajlirate u paket umeto u DLL.

Drugo rešenje ovog problema je upotreba paketa samo za vreme izvršavanja kako za EXE tako i za DLL, tako da se nijedan deo koda ne duplira. Sporedan efekat je da će postojati samo jedan `Application` objekat koji dele program i DLL, umesto dva zasebna objekta, te nam više neće biti potreban kod za sinhronizaciju.

Drugo pojednostavljenje programa se dobija jer neprioritetni formular koji se nalazi unutar DLL-a može da komunicira sa glavnim formularom pristupanjem listi formulara (koja je na raspolaganju u deljenom globalnom `Screen` objektu) ili jednostavnom upotrebom svojstva `Application.MainForm`. U primeru FormDll ja sam promenio drugu rutinu koju izvozi DLL u jednostavniju verziju:

```
procedure ShowColor (Col: LongInt); stdcall;
```

Modifikovao sam kod uklanjanjem referenci na hendl formulara i poruke i ponovo sam napisao kod za kontrolu Apply, tako da koristi glavni formular umesto da šalje korisničku Windows poruku:

```
procedure TFormScroll.ApplyClick (Sender: TObject);
begin
  // access the main form directly
  Application.MainForm.Color := SelectedColor;
end;
```

Problem je što ukoliko sada želite da pokrenete ovaj kod (koji nije finalna verzija izvornog koda fajlova), ponašanje nije onakvo kakvo ste očekivali. Glavni formular i formular iz DLL-a uopšte nisu sinhronizovani, postoje dva elementa na Taskbaru i dalje postoje svi ostali problemi prve verzije primera FormDLL. Problem je u činjenici da kada pokrenete program, DLL se inicijalizuje pre aplikacije, te je on taj koji inicijalizuje Forms jedinicu VCL-a. U okviru DLL-a VCL kreira objekat `Application`, ali ne kreira odgovarajući prozor.

Postoje dva radikalno drugačija pristupa ovom problemu inicijalizacije: jedan je promena redosleda inicijalizacije dinamičkim učitavanjem DLL-a pošto započne izvršavanje aplikacije; drugi je dodavanje dodatnog inicijalizacionog koda programu.

Dinamičko učitavanje DLL-a sa paketima

Prvo rešenje je pokazano bibiliotekom FirstDLLD i primerom UseDyna, koji dinamički učitava DLL izrađen upotrebom paketa samo za vreme izvršavanja. Glavni program učitava DLL prilikom pokretanja u obradi događaja `OnCreate` formulara:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  hInstDll := LoadLibrary ('ForrnDllD.dll');
  if hInstDll <= 0 then
```

```
    raise Exception.Create ('FormDllD library not found');
end;
```

U programu nisam deklarisao funkcije koje DLL izvozi da bih izbegao implicitnu vezu bibilioteke. Umesto toga sam deklarisao dva tipa procedura:

```
type
  TGetColorProc function (Col: LongInt): LongInt; stdcall;
  TShowColorProc = procedure (Col: LongInt); stdcall;
```

Ovi tipovi se koriste za konvertovanje generičkih pokazivača koji se dobijaju funkcijom `GetProcAddress`, što smo već videli u primeru DynaCall:

```
procedure TForm1.BtnChangeClick(Sender: TObject);
var
  Col: LongInt;
  GetColorProc: TGetColorProc;
  FPointer: TFarProc;
begin
  FPointer := GetProcAddress (hInstDll, 'GetColor');
  if FPointer = nil then
    raise Exception.Create ('GerColor DLL function not found');
  GetColorProc := TGetColorProc (FPointer);
  // original code
  Col := ColorToRGB (Color);
  Color := GetColorProc (Col);
end;
```

Upotreba dinamičkog učitavanja je korektan pristup koji Delphi zvanično podržava. Ipak, funkcije morate dinamički pozivati, što zahteva nešto više kodiranja.

Popravljanje koda inicijalizacije

Alternativno rešenje je da spoljašnje funkcije budu definisane u glavnom programu i da omogućite da se DLL prvo pokrene i inicijalizujete VCL, a da VCL kreira objekat `Application` bez odgovarajućeg prozora. Zapravo, biblioteci možemo da dodamo jednu liniju koda da bismo zatražili kreiranje prozora objekta `Application` tokom procesa inicijalizacije biblioteke (pre nego što izvršni fajl kreira svoje glavne objekte). Ovo postižemo pisanjem koda u inicijalizacionom odeljku jedne od jedinica DLL-a:

```
initialization
  Application.CreateHandle;
```

Pošto se ovaj kod nalazi u DLL-u, aplikacija ne može da učita njegovu ikonu. Rešenje je, zapravo, veoma jednostavno. U obradi događaja `OnCreate` glavnog formulara (u glavnom programu) jednostavno ponovo učitajte ikonu:

```
procedure TForm1.FormCreate (Sender: TObject);
begin
  // reload the icon of the application
  Application.Icon.Handle :=
    LoadIcon (HInstance, 'MAINICON');
end;
```

Pretraživanje strukture paketa

Verovatno se pitate da li postoji način na koji možete da saznate da li je paket povezan u vreme dizajniranja ili se koristi kao paket samo za vreme izvršavanja. Ne samo da je ovo moguće u Delphiju, već imate mogućnost da pretražite sveukupnu strukturu aplikacije. Komponenta može da koristi nedokumentovanu globalnu promenljivu `ModuleIsPackage`, koja je deklarisana u jedinici SysInit. Ovo Vam nikada neće biti potrebno, ali je tehnički moguće da komponenta ima drugačiji kod u zavisnosti od toga da li je u paketu ili ne. Naredni kod izdvaja naziv paketa samo za vreme ivršavanja koji sadrži komponentu, ukoliko je ima:

```
var
  fPackName: string;
begin
  // get package name
  SetLength (fPackName, 100);
  if ModuleIsPackage then
  begin
    GetModuleFileName (HInstance,
      PChar (fPackName), Length (fPackName));
    fPackName := PChar (fPackName) // string length fixup
  end
  else
    fPackName := 'Not packaged';
```

Pored pristupanja informacijama paketa iz komponente (kao što je to učinjeno u prethodnom kodu), isto to možete da učinite iz specijalne tačke ulaza u paket, funkcije `GetPackageInfoTable`. Ova funkcija daje neke specifične informacije o paketu koje Delphi čuva kao resurse i uključuje u DLL. Na sreću, ne moramo da koristimo tehnike niskog nivoa da bismo došli do ovih informacija, jer Delphi obezbeđuje neke funkcije visokog nivoa pomoću kojih možemo da manipulišemo ovim informacijama.

U osnovi postoje dve funkcije koje možete koristiti za pristupanje informacijama paketa:

- `GetPackageDescription` daje string koji sadrži opis paketa. Da biste pozvali ovu funkciju, morate da obezbedite naziv modula (biblioteku paketa) kao jedini parametar.
- `GetPackageInfo` ne daje direktno informacije o paketu. Umesto toga Vi prosleđujete funkciju koju će pozvati za svaki element interne strukture podataka paketa. U praksi, funkcija `GetPackageInfo` će pozvati Vašu funkciju za svaku jedinicu koja se nalazi u paketu. Uz to, `GetPackageInfo` određuje nekoliko zastavica u promenljivoj Integer.

Ove dve funkcije nam omogućavaju da pristupimo internim informacijama o paketu, ali kako ćemo da znamo koje pakete naša aplikacija koristi? To se može odrediti pretraživanjem izvršnog fajla upotrebom funkcija niskog nivoa, ali Delphi Vam i tu pomaže jer obezbeđuje jednostavniji način. Funkcija `EnumModules` ne daje direktno informacije o modulima aplikacije već omogućava da joj prosledite funkciju koju ona poziva za svaki modul aplikacija, glavni izvršni fajl i za svaki od paketa na koji se aplikacija oslanja.

Da bih pokazao ovaj pristup, ja sam izradio program koji prikazuje modul i informaciju o paketu u komponenti TreeView. Svaki čvor prvog nivoa odgovara modulu, a u okviru svakog modula ja sam izradio poddrvo koje prikazuje sadržane i potrebne pakete za taj modul, kao i opis paketa i zastavice kompajlera (`RunOnly` i `DesignOnly`). Izlaz ovog primera možete videti na slici 14.10.

SLIKA 14.10 *Izlaz primera PackInfo, sa detaljima paketa koje koristi*

Pored komponente TreeView, ja sam glavnom formularu dodao i nekoliko drugih komponenata, ali sam ih sakrio tako da se ne prikazuju: DBEdit, Chart i FilterComboBox. Ove pakete sam dodao samo da bih u aplikaciju uključio što više paketa samo za vreme izvršavanja, pored univerzalnog paketa VLC50.DPL. Jedini metod klase formulara je `FormCreate`, koji poziva funkciju za enumeraciju modula:

```
procedure TForm1.FormCreate (Sender: TObject);
begin
  EnumModules (ForEachModule, nil);
end;
```

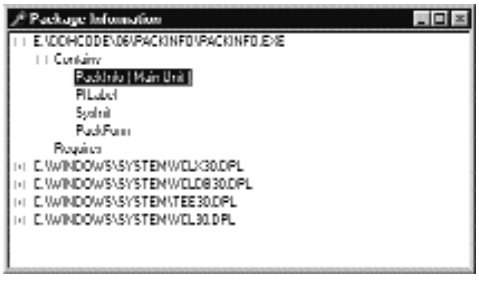
Funkcija `EnumModules` prihvata dva parametra. Prvi je funkcija (u našem slučaju `ForEachModule`), a drugi je pokazivač na strukturu podataka koju će koristiti funkcija (u našem slučaju `nil`, jer nam nije potrebna). Funkcija mora da prihvati dva parametra, vrednost `HInstance` i pokazivač bez tipa, i mora dati `Boolean` vrednost. Funkcija `EnumModules` će pozvati našu funkciju za svaki modul, prosleđujući hendl instance svakog modula kao prvi parametar i pokazivač strukture podataka (`nil` u našem primeru) kao drugi parametar:

```
function ForEachModule (HInstance: Longint;
  Data: Pointer): Boolean;
var
  Flags: Integer;
  ModuleName, ModuleDesc: string;
  ModuleNode: TTreeNode;
begin
```

```
  with Form1.TreeView1.Items do
  begin
    SetLength (ModuleName, 200);
    GetModuleFileName (HInstance,
      PChar (ModuleName), Length (ModuleName));
    ModuleName := PChar (ModuleName);  // fixup
    ModuleNode := Add (nil, ModuleName);

    // get description and add fixed nodes
    ModuleDesc := GetPackageDescription (PChar (ModuleName));
    ContNode := AddChild (ModuleNode, 'Contains');
    ReqNode := AddChild (ModuleNode, 'Requires');

    // add information if the module is a package
    GetPackageInfo (HInstance, nil,
      Flags, ShowlnfoProc);
    if ModuleDesc <> '' then
    begin
      AddChild (ModuleNode,
      'Description: ' + ModuleDesc):
    if Flags and pfDesignOnly = pfDesignOnly then
      AddChild (ModuleNode, 'Design Only');
    if Flags and pfRunOnly = pfRunOnly then
      AddChild (ModuleNode, 'Run Only');
    end;
  end;
  Result := True;
end;
```

Kao što možete videti iz prethodnog koda, funkcija `ForEachModule` započinje dodavanjem naziva modula kao glavnog čvora drveta (pozivanjem metoda `Add` objekta `TreeView1.Items` i prosleđujući `nil` kao prvi parametar). Zatim dodaje dva fiksna dete-čvora, koja se čuvaju u promenljivima `ContNode` i `ReqNode` deklarisanim u odeljku `implementation` ove jedinice.

Zatim, program poziva funkciju `GetPackageInfo` i prosleđuje drugu funkciju, `ShowInfoProc`, koju ću ubrzo objasniti. Program dodaje detalje za glavni modul (videti sliku 14.11), jer će to obezbediti listu jedinica aplikacije. Na kraju ove funkcije dodajemo još informacija ukoliko je modul paket, kao što je njegov opis i zastavice kompajlera (mi znamo da je to paket ukoliko opis nije prazan string).



SLIKA 14.11 *Primer PackInfo takođe prikazuje jedinice koje su deo aktuelnog projekta*

Ranije sam pomenuo prosleđivanje još jedne funkcije, procedure `ShowInfoProc`, funkciji `GetPackageInfo`, koja odmah poziva našu funkciju za svaki sadržani ili neophodni paket modula. Ova procedura kreira string koji opisuje paket i njegove glavne zastavice (dodaju se unutar zagrada), a zatim taj string umeće ispod jednog od dva čvora (`ContNode` i `ReqNode`), već prema tipu modula. Tip modula možemo da odredimo proverom parametra `NameType`. Evo kompletnog koda naše druge funkcije:

```
procedure ShowInfoProc (const Name: string;
  NameType: TNameType; Flags: Byte; Param: Pointer);
var
  FlagStr: string;
begin
  FlagStr := ' ';
  if Flags and ufMainunit <> 0 then
    FlagStr  FlagStr + 'Main Unit ';
  if Flags and ufPackageUnit <> 0 then
    FlagStr := FlagStr + 'Package Unit ';
  if Flags and ufWeakUnit <> 0 then
    FlagStr := Flagstr + 'Weak Unit ';
  if Flagstr <>  ' ' then
    Flagstr := ' (' + FlagStr + ')';
  with Form1.TreeView1.Items do
    case NameType of
      ntContainsUnit:
        AddChild (ContNode, Name + FlagStr);
      ntRequiresPackage:
        AddChild (ReqNode, Name);
    end;
end;
```

Primetićete da parametar `Flags` ne sadrži stil informacije zastavice, kako to nagoveštava help. Ukoliko želite da istražite ovu karakteristiku, proučite jedinicu SysUtils.

Šta je sledeće?

U ovom poglavlju smo videli kako možemo da pozivamo funkcije koje se nalaze u DLL-ovima kreiranim u jeziku C++ ili drugim jezicima, kako da kreiramo DLL-ove upotrebom Delphija i kako da koristimo stringove i smestimo Delphi formulare unutar biblioteke. DLL-ovi su bili jedan od tradicionalnih pristupa pisanju aplikacija upotrebom više programskih jezika i okruženja. Danas COM i OLE obezbeđuju naprednije tehnike.

Druga tehnika za izvoženje objekata iz DLL-ova je upotreba paketa, mada postoje mnoge teme vezane za ovu tehniku. Kada razmišljate o DLL-ovima i drugim tehnikama, imajte na umu da, mada Delphi i Windows dele mnoge elemente, oni imaju i različite poglede na programiranje. Kada god je moguće, poštujte Delphi objektno orijentisani pristup, a ne Windows proceduralni pristup, i verovatno ćete od toga imati koristi. Zapravo, prednosti objektne orijentacije su toliko važne da je Microsoft predstavio neke objektno orijentisane koncepte direktno u svom sistemu. Ukratko, OLE i COM su primeri ovih ugrađenih tehnika.

Naredna poglavlje su u potpunosti posvećena ovim temama: COM i OLE, OLE Automation, OLE Documents i ActiveX kontrole.

POGLAVLJE

15

COM programiranje

Prema Microsoftu, COM tehnologija će imati osnovnu ulogu u evoluciji Windows platforme. Microsoft je prvobitno koristio termin OLE koji se odnosio na ovu tehnologiju, zatim je mnogo češće počeo da koristi termin COM, a trenutnu verziju naziva COM+. COM nije samo jedna tehnologija, već osnovna infrastruktura operativnog sistema, koja se primenjuje u mnogim situacijama. Ovo poglavlje pokazuje da je COM programiranje verovatno jednostavnije nego što pretpostavljate. U ovom poglavlju izradićemo naš prvi COM objekat i integrisaćemo naše COM objekte u Windows ljusku (shell). Biblioteke tipova, Automation i druge teme će biti objašnjene u narednom poglavlju. Ja ću se držati ovih osnovnih elemenata da bih Vam omogućio da shvatite ulogu ove tehnologije i neću se upuštati u detalje.

Šta je OLE, a šta je COM?

Deo zabune do koje dolazi je činjenica da je Microsoft menjao nazive iz marketinških razloga. Sve je počelo od naziva Object Linking and Embedding (OLE — povezivanje i ugnežđavanje objekata), što predstavlja proširenje DDE modela (Dynamic Data Exchange — dinamička razmena podataka). Upotreba Clipboarda Vam omogućava da kopirate sirove podatke, a upotreba DDE-a Vam omogućava da povežete delove dva dokumenta. Object Linking and Embedding (OLE) Vam omogućava da kopirate podatke iz server aplikacije u klijent aplikaciju, uz neke informacije koje se tiču servera, ili da se pozivate na neke informacije koje se čuvaju u Windows Registryju. Sirovi podaci se mogu kopirati uz link (Object Embeding) ili se mogu čuvati u originalnom fajlu (Object Linking). Object Linking and Embedding dokumenata je kasnije nazvano *OLE Documents*, a sada se naziva Active Documents (aktivni dokumenti).

Microsoft je OLE unapredio u OLE2 i počeo je da dodaje nove karakteristike, kao što su OLE Automation i OLE Controls. Sledeći korak je bio izrada Windows 95 ljuske upotrebom OLE tehnologije i interfejsa, a zatim je OLE Controls (prethodno su bile poznate i kao OCX) preimenovao u ActiveX kontrole, menjajući specifikaciju da bi se omogućila lakša distribucija preko Interneta.

Kako je ova tehnologija proširena i kako je postala sve više važna za Windows platformu, Microsoft je naziv promenio u OLE, zatim u COM, a sada u COM+ za Windows 2000. Ove promene naziva se samo delimično odnose na promenu tehnologije i više su proizvod marketinških potreba.

Šta je, dakle, COM? U osnovi, Component Object Model (model objekata komponenata), ili COM, je tehnologija koja definiše standardni način komunikacije između klijent modula i server modula preko specifičnog interfejsa. Ovde "modul" označava aplikaciju ili biblioteku (DLL); dva modula se mogu izvršavati na istom kompjuteru ili na različitim kompjuterima koji su povezani preko mreže. Mogući su mnogi interfejsi, u zavisnosti od uloge klijenta i servera, a možete da dodate i nove interfejse za specifične namene. Ovi interfejsi su implementirani preko server objekata. Server objekat obično implementira više od jednog interfejsa, a svi server objekti imaju nekoliko zajedničkih mogućnosti, jer svi moraju da implementiraju IUnknown interfejs.

Dobra vest je da Delphi potpuno podržava COM. Kada pogledate izvorni kod, Object Pascal je, izgleda, lakše koristiti od C++ ili drugih jezika za pisanje COM objekata. Ova jednostavnost je uglavnom proizvod inkorporacije tipova interfejsa u jezik Object Pascal. Usput, interfejsi se, takođe, na sličan način koriste za integraciju Java COM-a na Windows platformi.

Namena COM interfejsa je komunikacija između modula softvera, dva izvršna fajla ili izvršnog fajla i DLL-a. Implementiranje COM objekata u DLL-ovima je, uopšte uzev, jednostavnije, jer pod platformom Win32 program i DLL koji on koristi nalaze se u istom memorijskom adresnom prostoru. To znači da ukoliko program prosledi memorijsku adresu DLL-u, adresa ostaje validna. Kada koristite dva izvršna fajla, COM treba da obavi mnogo posla da bi omogućio da dve aplikacije komuniciraju. Ovaj mehanizam se naziva uređivanje (marshaling). Primetićete da je DLL koji implementira COM objekte opisan kao server u procesu (in-process), a kada je server zaseban fajl, naziva se server van procesa (out-of-proces). Ipak, kada se DLL-ovi izvršavaju na drugoj mašini (DCOM) ili unutrašnjem okruženju (MTS), takođe se nalaze van procesa.

Implementiranje IUnknown

Pre nego što pogledamo primer COM programiranja, želeo bih da Vam predstavim osnove COM-a. Prvo, svaki COM objekat mora da implementira `IUnknown` interfejs. Ovo je osnovni interfejs iz koga se izvodi svaki Delphi interfejs, a Delphi obezbeđuje nekoliko različitih klasa sa implementacijama `IUnkown` spremnim za upotrebu, uključujući `TInterfacedObject` i `TComObject`. Prva klasa može da se koristi za interni COM objekat, dok druga mora da se koristi za izvoženje objekta sa servera. Kao što ću navesti u Poglavlju 16, nekoliko drugih klasa se nasleđuje iz klase `TComObject` i obezbeđuje podršku za još interfejsa koji su potrebni za Automation servere ili ActiveX kontrole.

Interfejs `IUnknown` sadrži tri metoda: `Add`, `Release` i `QueryInterface`. Evo definicije interfejsa `IUnknown` (izvučen je iz jedinice System):

```
type
  IUnknown = interface
    ['{00000000-0000-0000-C000-000000000046}']
  function QueryInterface(const IID: TGUID;
    out Obj): Integer; stdcall;
  function SddRef: Integer; stdcall;
  function Release: Integer; stdcall;
end;
```

Metodi `_AddRef` i `_Release` se koriste za implementiranje brojanja referenci. Metod `QueryInterface` prikazuje informacije tipa i kompatibilnost tipa objekata.

NAPOMENA

U prethodnom kodu možete videti primer parametra `out`, parametra koji se dobija od metoda kojeg poziva program, ali bez inicijalne vrednosti koju prosleđuje program koji poziva metod. Parametri `out` su dodati Delphijevom jeziku Object Pascal radi podrške za COM. Takođe je važno primetiti da dok je Delphijeva definicija jezika za tip interfejsa dizajnirana radi komapatibilnosti sa COM-om, Delphi interfejsi ne zahtevaju COM. Ovo je već istaknuto u Poglavlju 3, u kome sam izradio složeni primer na osnovu interfejsa bez ikakve COM podrške. ■

Obično nije potrebno da implementirate ove metode, jer ih možete naslediti iz neke od Delphi klasa koje ih već podržavaju. Najvažnija klasa je `TComObject`, definisana u jedinici ComObj. Kada izrađujete COM server, u opštem slučaju ćete nasleđivati iz ove klase. Pošto je `TComObject` složena klasa, sledeći isečak prikazuje samo ključne elemente:

```
type
  TComObject = class(TObject, IUnknown, ISupportErrorInfo)
  private
    FNonCountedObject: Boolean;
    FRefCount: Integer;
  protected
    { IUnknown }
    function tUnknown.QueryInterface = CbjQueryInterface;
    function tUnknown._AddRef = ObjAddRef;
    function Ibnknown._Release = ObjRelease;
    { ISupportfrrorlnfo }
    function InterfaceSupportsErrorInfo(const iid: TIID):
      HResult; stdcall;
  public
```

```
    constructor Create;
    destructor Destroy; override;
    procedure Initialize; virtual;
    function ObjAddRef: Integer; virtual; stdcall;
    function ObjQueryInterface(const IID: TGUID; out Obj): HResult;
      virtual; stdcall;
    function ObjRelease: Integer; virtual; stdcall;
    property RefCount: Integer read FRefCount;
  end;
```

Ova klasa implementira interfejs `IUnknown` (upotrebom metoda `ObjAddRef`, `ObjQueryInterface` i `ObjRelease`, što je naznačeno iskazima mapiranja metoda u zaštićenom odeljku klase) i interfejs `ISuppportedErrorInfo` (preko metoda `InterfaceSupportsErrorInfo`). Implementacija prebrojavanja referenci za klasu `TComObject` je proširena radi podrške linija. Umesto upotrebe funkcija `Inc` i `Dec`, kod koristi API funkcije InterlockedIncrement i `InterlockedDecrement`, kao što možete videti u ivornom kodu klase:

```
function TComObject.ObjAddRef: Integer;
begin
  Result := InterlockedIncrement(FRefCount);
end;

function TComObject.ObjRelease: Integer;
begin
  Result := InterlockedDecrement(FRefCount);
  if Result = 0 then Destroy;
end;
```

Kao što možete videti, implementacija metoda `Release` uklanja objekat kada nema više referenci na njega. Na prvi pogled, potreba za pozivanjem ovog metoda svaki put kada operišete objektom izgleda kao mnogo posla. Ipak, možda ste zapamtili iz Poglavlja 3 da, kada koristite promenljive interfejsa za referisanje na objekte, Delphi automatski dodaje prebrojavanje poziva kompajliranom kodu, koji automatski uklanja objekat na koji nema reference. Ova karakteristika koja Vas oslobađa posla čini Delphi pogodnim alatom za COM programiranje.

Najsloženiji metod je `QueryInterface`, koji je u Delphiju implementiran preko metoda `GetInterface` klase `TObject`:

```
function TComobject.ObjQueryInterface(
  const lID: TGLJID; out Obj): HResult;
begin
  if GetInterface(IID, Obj) then
    Result := S_OK
  else
    Result := E_NOINTERFACE;
end;
```

Uloga metoda `QueryInterface` je dvostruka:

- `QueryInterface` se koristi za proveru tipa. Program može pitati objekat sledeće: Da li si tip objekta koji me interesuje? Da li implementiraš interfejs koji želim da pozovem? A specifične metode? Ukoliko je odgovor ne, program može da potraži drugi objekat, možda i drugi server.

- Ukoliko je odgovor da, `QueryInterface` obično daje pokazivač na objekat, koristeći parametar reference i izlazni parametar (`out`).

Da biste razumeli ulogu metoda `QueryInterface`, važno je da imate na umu da COM objekat može da implementira više interfejsa, kao što to čini `TComObject`. Kada pozovete `QueryInterface`, možete da zatražite jedan od mogućih interfejs objekata upotrebom parametra `TGUID`.

Globalno jedinstveni identifikatori

Metod `QueryInterface` sadrži specijalan parametar tipa `TGUID`. To je ID koji identifikuje bilo koju COM server klasu i bilo koji interfejs sistema. Kada želite da znate da li objekat podržava određeni interfejs, pitaćete objekat da li implementira interfejs sa datim ID-om (koji za unapred određene OLE interfejse određuje Microsoft).

Drugi ID se koristi za označavanje specifične klase, specifičnog servera. Windows Registry čuva ovaj ID, sa oznakama odgovarajućeg DLL-a ili izvršnog fajla. Programeri OLE servera određuju identifikator klase.

Oba ova ID-a su poznata kao GUID-ovi, ili globalno jedinstveni identifikatori (Globally Unique IDentifiers). Ukoliko svaki programer koristi broj da bi označio sopstvene OLE servere, kako može biti siguran da se ove vrednosti neće duplirati? Kratak odgovor je da ne zna. Pravi odgovor je da je GUID toliko veliki broj (sa 16 bajtova, ili 128 bita, ili broj sa 38 cifara) tako da je statistički nemoguće da se dođe do dva slučajna broja koja imaju istu vrednost. Dalje, programeri bi trebalo da koriste specifičan API poziv `CoCreateGuid` (direktno ili preko njihovih razvojnih okruženja), da bi došli do validnog GUID-a koji odslikava neke sistemske informacije.

Zapravo, GUID-ovi koji su kreirani na mašinama sa mrežnim karticama su sasvim sigurno jedinstveni, jer mrežne kartice sadrže jedinstvene serijske brojeve koji čine osnovu za kreiranje GUID-a. GUID-ovi kreirani na mašinama sa CPU ID-ovima (kao što je Pentium III) trebalo bi da su takođe sasvim sigurno jedinstveni, čak i kada nema mrežne kartice. Kada nema jedinstvenog identifikatora hardvera, teško da se GUID-ovi mogu duplirati.

UPOZORENJE

Pored toga što morate da pazite da ne kopirate GUID iz nečijeg programa (koji kao rezultat može dati dva potpuno različita COM objekta koji korise isti GUID), nikada ne treba da sami načinite ID unošenjem nasumičnog niza brojeva. Windows proverava ID-ove, a upotrebom nasumičnog niza brojeva nećete dobiti validan ID. OLE server koji nema validan ID se ne prepoznaje, i nećete dobiti poruku sa greškom! Windows, takođe, neće uključiti API ili tehniku za proveru GUID-a. Rizik ručnog kreiranja ID-a klasa ili ID interfejsa je da možete slučajno da duplirate GUID koji se već koristi negde u okviru sistema. Ipak, da biste izbegli ovaj problem, jednostavno prititsnite kombinaciju tastera Ctrl+Shift+G u Delphi editoru i dobićete novi, pravilno definisani GUID. ■

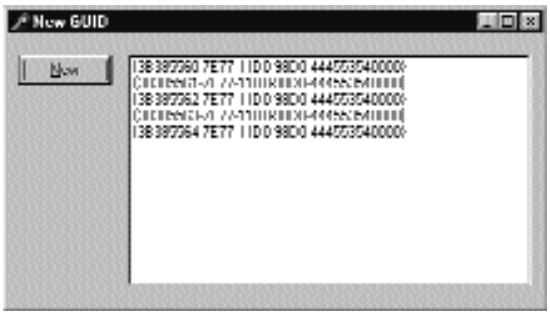
Delphi definiše tip podataka `TGUID` (u jedinici System) za čuvanje ovakvih brojeva:

```
type
  TGUID = record
    D1: Integer;
    D2: Word;
    D3: Word;
    D4: array [0..7] of Byte;
end;
```

Ova struktura je prilično čudna, ali je Windows zahteva kao takvu. Možete dodeliti vrednost GUID-u upotrebom standardne heksadecimalne notacije, kao u narednom isečku koda:

```
const
  Class_ActiveForm1: TGUID =
    '{1AFA6D61-7B89-98D0-444553540000}';
```

Ukoliko je potrebno da GUID kreirate ručno i ne u Delphi okruženju, možete jednostavno da pozovete Windows API funkciju `CoCreateGuid`, što je pokazano primerom NewGuid (videti sliku 15.1). Ovaj primer je toliko jednostavan, da sam odlučio da ga ovde ne prikažem listingom. (Izvorni kod ove aplikacije možete pronaći uz ostale primere Poglavlja 15 u direktorijumu koji ste preuzeli sa Sybexovog web sajta.)



SLIKA 15.1 *Primer GUID-ova koje je generisao primer NewGuid. Vrednosti zavise od mog kompjutera i vremena kada sam pokrenuo program*

Za rukovanje GUID-ovima Delphi obezbeđuje funkciju `GUIDToString` i suprotnu funkciju `StringToGuid`. Takođe, možete da koristite Windows API funkcije, kao što je `StringFormGuid2`, ali u ovom slučaju morate da koristite tip WideString umesto tipa string. Svaki put kada se koristi OLE, morate da koristite tip WideString, izuzev ukoliko koristite Delphi funkcije koje automatski obavljaju potrebnu konverziju. Zapravo, OLE API funkcije koriste tip `PWChar` (pokazivač na nizove karaktera koji se završavaju null vrednošću), ali jednostavna konverzija WideString u `PWChar` će obaviti posao.

SAVET

Imajte na umu da GUID-ovi postoje u raznim oblicima. Najvažniji tipovi su ID-ovi interfejsa (ili IID), koji se odnose na interfejs, i ID-ovi klasa (ili CLSID), koji se odnose na specifični objekat servera. Ove dve vrste ID-ova koriste GUID stil. ■

Uloga radionica klasa (Class Factories)

Kada registrujemo GUID COM objekta u Registryju, možemo da koristimo specifične API funkcije da bismo kreirali objekat, kao što je API funkcija `CreateComObject`:

```
function CreateComObject (const ClassID: TGUID): IUnknown;
```

Ova API funkcija će pretražiti Registry, pronaći server koji registruje objekat sa datim GUID-om, učitati ga i, ukoliko je server DLL, pozvati metod DLL-a `DLLGetClassObject`. To je funkcija koju svaki server u procesu mora da obezbedi i izveze:

```
function DllGetClassObject (const CLSID, IID: TGUID;
  var Obj): HResult; stdcall;
```

Ova API funkcija kao parametar dobija zahtevanu klasu i interfejs, i kao rezultat daje objekat u parametru reference. Objekat dobijen ovom funkcijom je radionica klase (class factory).

Sada, šta je radionica klase? Kao što naziv sugeriše, radionica klase je objekat koji može da kreira druge objekte. Svaki server može da ima više objekata. Server prikazuje radionicu klase, a radionica klase može da kreira jedan od ovih različitih objekata. Svaki objekat, zatim, može da ima veliki broj instanci. Jedna od mnogih prednosti Delphijevog pojednostavljenog pristupa COM programiranju je da sistem može da nam obezbedi radionicu klase. Iz ovog razloga ja neću dodati radionicu klase našem jednostavnom primeru.

Poziv API funkcije `CreateComObject` se ne zaustavlja prilikom kreiranja radionice klase. Posle dobijanja radionice klase, `CreateComObject` poziva metod `CreateInstance` interfejsa `IClassFactory`. Ovaj metod kreira zahtevani objekat i daje ga nama. Ukoliko se ne dese greške, ovaj objekat postaje vrednost koja se dobija API funkcijom `CreateComObject`.

Podešavanjem ovog mehanizma (uključujući radionicu klase i poziv `DLLGetClassObject`) kreiranje objekata činite veoma jednostavnim. Ono što je sjajno u Delphiju jeste to da složenim mehanizmom rukuje sistem koji se izvršava. Dakle, vreme je da se detaljno upoznamo sa tim kako Delphi čini COM veoma lakim.

Radionice klasa i druge Delphi COM klase

Pored klase `TComObject`, Delphi sadrži nekoliko drugih unapred određenih COM klasa. Mi ćemo ih koristiti u narednim odeljcima, ali ovde je data lista najvažnijih COM klasa Delphi VCL-a:

- Klasa `TInterfacedObject`, definisana u jedinici System, izvedena iz klase `TObject` i implementira interfejs `IUnknown`. Koristi se samo za interne objekte.
- Klasa `TComObject`, definisana u jedinici `ComObj`, izvedena iz klase `TObject` i implementira interfejs `IUnknown` i interfejs `ISupportErrorInfo`. Za razliku od klase `TInterfacedObject`, ova klasa takođe ima odgovarajuću radionicu klase.
- Klasa `TTypedComObject`, definisana u jedinici ComObj, izvedena iz klase TObject i implementira interfejs `IProvideClassInfo` (uz interfejse `IUnknown` i `ISupportErrorInfo` koji su već implementirani u osnovnoj klasi, klasi `TComObject`).
- Klasa `TAutoObject`, definisana u jedinici ComObj, izvedena iz klase `TTypedComObject` i takođe implementira interfejs `IDispatch`.
- Klasa `TActiveXControl`, definisana u jedinici AxCtrls, izvedena iz klase `TAutoObject` i implementira brojne interfejse (neki od njih su `IPersistStreamInit`, `IPersistStorage`, `IOleObject` i `IOleControl`).

Za svaku od ovih klasa Delphi, takođe, definiše radionicu klase. Klase radionice klase formiraju drugu hijerarhiju sa istom strukturom. Nazivi su `TComObjectFactory`, `TTypedComObjectFactory`, `TAutoObjectFactory` i `TActiveXControlFactory`. Radionice klasa su važne i potrebne su svakom COM serveru. Obično radionice klasa koristimo kreiranjem objekta u inicijalizacionom odeljku definišući odgovarajuću klasu objekta servera.

Prvi COM server

Ne postoji bolji način da shvatite COM server nego da izradite jednostavan COM server koji se nalazi u DLL-u. Biblioteka koja sadrži COM objekat je u Delphiju označena kao ActiveX biblioteka. Stoga možemo da započnemo programiranje ovog projekta ukoliko odaberemo File➥New, zatim pređemo na stranu ActiveX i odaberemo opciju ActiveX Library. Na ovaj način se generiše fajl projekta koji sam ja sačuvao pod nazivom FirstCom. Ovo je kompletan izvorni kod:

```
library FirstCom;

uses
  ComServ;

exports
  DllGetClassObject,
  DllCanUnloadNow,
  DllRegisterServer,
  DllUnregisterServer;

{$R *.RES}

begin
end.
```

Četiri funkcije koje su izvezene DLL-om su neophodne za COM usluge i sistem ih koristi na sledeći način:

- za pristupanje biblioteci klase (`DllGetClassObject`);
- za proveru da li je server uklonio sve svoje objekte i da li se može izbaciti iz memorije (`DllCanUnloadNow`);
- za dodavanje ili uklanjanje informacija o serveru u Windows Registryju (`DllRegisterServer` i `DllUnregisterServer`).

U opštem slučaju ne morate da implementirate ove funkcije jer Delphi obezbeđuje osnovnu implementaciju u jedinici ComServ. Zbog toga ih u kodu našeg servera treba samo izvesti.

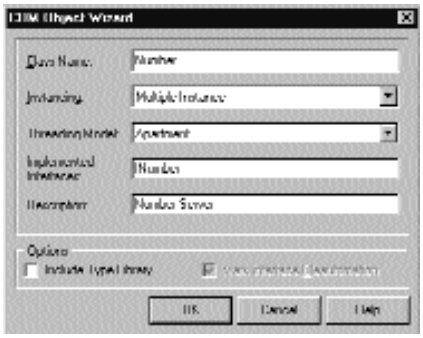
COM interfejsi i objekti

Sada kada imamo strukturu našeg COM servera na pravom mestu, možemo da počnemo programiranje Com servera. Prvi korak je pisanje koda interfejsa koji želimo da implementiramo u server. Interfejsi mogu biti veoma slični kodu apstraktne klase, prikazujući sve metode koje želimo da učinimo dostupnim sa našeg servera. (Ja sam već razmatrao Object Pascal interfejse u Poglavlju 3.) Ovde navodim kod jednostavnog interfejsa koji bi trebalo da dodate u zasebnu jedinicu (u našem primeru je nazvana NumIntf):

```
type
  INumenr = interface
    ['{B4131140-7C2F-11D0-98D0-444553540000}']
    function GetValue: Integer; stdcall;
    procedure SetValue (New: Integer); stdcall;
    procedure Increase; stdcall;
end;
```

IID se dodaje kodu kada se pritisne kombinacija tastera Ctrl+Shift+G.

Posle deklarisanja korisničkog interfejsa, serveru možemo da dodamo objekat. Da bismo ovo postigli, možemo da upotrebimo COM Object Wizard (do kojeg se može doći sa strane ActiveX okvira za dijalog File➥New). Okvir za dijalog ovog čarobnjaka možete videti na slici 15.2. Ovde treba da unesete naziv klase servera, interfejs koji želite da implementirate i opis. Ja sam onemogućio generisanje biblioteke tipa da bih izbegao uvođenje previše tema odjednom. Takođe bi trebalo da odaberete modele instanciranja i više linija, kao što je opisano u dodatku.



SLIKA 15.2 *COM Object Wizard*

Kod koji generiše COM Object Wizard je, zapravo, veoma jednostavan. Interfejs sadrži definiciju klase koju treba ispuniti metodima i podacima:

```
type
  TNumber = class (TComObject, INumber)
    protected
      {Declare INumber methods here}
end;
```

Klasa servera je izvedena iz klase `TComObject`, koju sam razmatrao u poslednjem odeljku. U kodu koji je generisao čarobnjak, posle klase servera dolazi definicija GUID-a za server:

```
const
  Class_Number: TGUID =
    ' {5B2EF181-3AAE-11D3-B9F1-00000100A27B} ';
```

Na kraju, postoji kod u odeljku `initialization` (koji koristi većinu opcija koje smo podesili u okviru za dijalog čarobnjaka):

```
initialization
  TComObjectFactory.Create (ComServer, TNumber,
  Class_Number, 'Number', 'Number Server',
  ciMultiInstance, tmApartment);
```

Ovaj kod kreira objekat klase `TComObjectFactory`, koji kao parametar prosleđuje globalni objekat `ComServer`, referencu klase na klasu koju smo upravo definisali, GUID klase, naziv servera, opis servera i modele instanciranja i više linija koje želimo da koristimo.

Globalni objekat `ComServer`, definisan u jedinici `ComServ`, upravlja radionicama klasa koje su na raspolaganju u biblioteci servera. Ovaj objekat koristi sopstveni metod `ForEachFactory` za pronalaženje klasa koje podržavaju dati zahtev za COM objektom, i beleži broj alociranih objekata. Kao što smo već videli, jedinica ComServ implementira funkcije koje su potrebne da DLL bude COM biblioteka.

Kada smo proučili kod koji je generisao čarobnjak, možemo da ga kompletiramo tako što ćemo klasi `TNumber` dodati metode koji su neophodni za implementiranje interfejsa `INumber`. Prvo napišite deklaraciju metoda:

```
type
  TNumber = class (TComObject, INumber)
  private
    fValue: Integer;
  public
    function GetValue: Integer; virtual; stdcall;
    procedure SetValue (New: Integer); virtual; stdcall;
    procedure Increase; virtual; stdcall;
end;
```

U ovom trenutku jednostavno aktivirajte kompletiranje klase tako što ćete pritisnuti kombinaciju tastera Shift+Ctrl+C i metode popunite odgovarajućim kodom. Ovo je toliko direktan postupak, da ga ovde neću prikazati; izvorni kod možete pronaći u direktorijumu FirstCom.

MODELI COM INSTANCIRANJA I VIŠE LINIJA

Kada kreirate COM server, potrebno je da odaberete odgovarajući model instanciranja i višelinijski model, koji značajno mogu uticati na ponašanje COM servera.

Instanciranje utiče samo na servere van procesa (bilo koji COM server koji se nalazi u zasebnom izvršnom fajlu, a ne u DLL-u) i može pretpostaviti tri vrednosti:

- Multiple označava da će sistem inicirati više instanci servera kada više klijent aplikacija zahteva COM objekat.
- Single označava da će postojati samo jedna instanca server aplikacije, čak i kada nekoliko klijent aplikacija zahteva COM objekat; kreira se više internih objekata da bi se obradili zahtevi.
- Internal označava da se objekat može kreirati samo unutar servera; klijent aplikacije ne mogu zahtevati objekat.

Druga odluka se odnosi na višelinijsku podršku COM objektu, što je validno samo za servere u procesu (DLL-ove). Višelinijski model je zajednička odluka klijent i server aplikacije: ukoliko se obe strane slože oko modela, on se koristi za povezivanje. Ukoliko se nađe zajedničko rešenje, COM će ipak moći da ostvari vezu upotrebom prilagođavanja, što može da uspori operacije. Takođe, imajte na umu da server ne samo da mora da prikaže višelinijski model u Registryju (što je rezultat podešavanja opcije čarobnjaka), već i u kodu mora da poštuje pravila višelinijskog modela. Ovde prikazujem najvažnije detalje različitih višelinijskih modela:

- Model single označava da nema prave višelinijske podrške. Zahtevi koji stižu do COM servera se smeštaju u red, tako da klijent može da obavi operacije jednu po jednu.
- Model apartment, ili single-threaded apartment, označava da samo proces koji je kreirao objekat može da poziva metode objekta. To znači da se zahtevi za svaki server objekat smeštaju u red, ali drugi objekti istog servera istovremeno mogu da primaju zahteve. Iz ovog razloga server objekat mora da preduzme dodatne korake zaštite prilikom pristupanja globalnim podacima servera (upotrebom kritičnih odeljaka, muteksa — mutex — ili nekih drugih tehnika sinhronizacije koje opisujem u Poglavlju 17). Ovo je višelinijski model koji obično koriste ActiveX kontrole unutar Internet Explorera.
- Model Free, ili multithreaded apartment, označava da nema restrikcije za klijente, što znači da više procesa istovremeno može da koristi isti objekat. Zbog toga svaki metod svakog objekta mora da zaštiti sebe i globalne podatke koje koristi od drugih poziva. Ovaj višelinijski model je složeniji za podršku servera od modela single i apartment, jer se čak i pristupanje instancama samog objekta mora obaviti veoma pažljivo.
- Poslednja opcija, Both, označava da server objekat podržava i model apartment i model free.

Inicijalizovanje COM objekta

Ukoliko se vratite na definiciju klase `TcomObject`, primetićete da sadrži konstruktor koji nije virtuelan. (Zapravo, sadrži više konstruktora koji nisu virtuelni, koje sam ja izostavio iz listinga.) Svaki `TComObject` konstruktor poziva virtuelni metod `Initialize`. Zbog toga, ukoliko želite da prilagodite kreiranje objekta, a zatim ga inicijalizujete, ne treba da kreirate novi konstruktor (koji se nikada neće pozvati). Ono što treba da učinite je da zaobiđete metod `Initialize` objekta, kao što sam ja učinio u klasi `TNumber`. Ovde dajem konačnu verziju ove klase:

```
type
  TNumber = class (TComObject, INumber)
  private
    fValue: Integer;
  public
    function GetValue: Integer; virtual; stdcall;
    procedure SetValue (New: Integer); virtual; stdcall;
    procedure Increase; virtual; stdcall;
    procedure Initialize; override;
    procedure Destroy; override;
end;
```

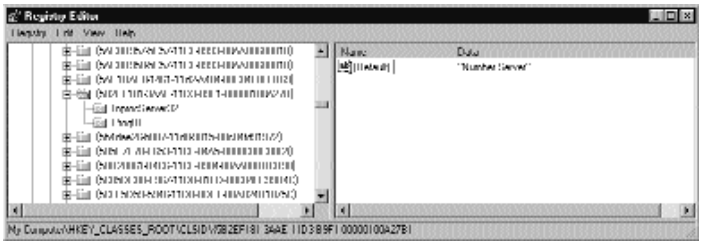
Kao što možete da vidite, ja sam takođe zaobišao destruktor klase jer sam želeo da testiram automatsko uklanjanje COM objekata koje obezbeđuje Delphi. Evo koda za ovaj pseudokonstruktor i destruktor:

```
procedure TNumber.Initialize;
begin
  inherited;
  fValue := 10;
end;

destructor TNumber.Destroy;
begin
  inherited;
  MessageBox (0, 'Object Destryed',
    'TDLLNumber', mb_OK); // API call
end;
```

U prvom metodu, pozivanje izvedene vezije je dobra praksa, iako metod `TComObject.Inititalize` ne sadrži nikakav kod u ovoj verziji Delphija. Destruktor, umesto toga, mora da pozove verziju osnovne klase. Ovo je kod koji je neophodan da bi naš COM objekat pravilno funkcionisao i da bi nas obavestio kada je objekat zaista uklonjen.

Testiranje COM servera

Sada kada smo završili pisanje COM server objekta, možemo da ga registrujemo i upotrebimo. Da biste registrovali server, jednostavno kompajlirajte njegov kod, a zatim u Delphiju upotrebite komandu menija Run➥register ActiveX Server. Ovo činite da biste registrovali server na sopstvenoj mašini, a rezultati registrovanja su prikazani na slici 15.3.



SLIKA 15.3 *Novi registrovani server u Windows RegEditu*

Kada distribuirate ovaj server, potrebno je da ga instalirate na klijent kompjutere. Da biste to učinili, možete napisati REG fajl za instaliranje servera u Registry. Ipak, ovo nije najbolji pristup, jer server već sadrži funkciju koju možete aktivirati da biste registrovali server. Ovu funkciju može aktivirati Delphi okruženje, kao što smo već videli, ili se može aktivirati na nekoliko drugih načina:

- Microsoftovom programu `RegSvr32.exe` možete proslediti COM server DLL kao parametar sa komandne linije, a program `RegSvr32.exe` možete pronaći u direktorijumu Windows/System.
- Možete da upotrebiti sličan demo program `TRegSvr.exe` koji dobijate uz Delphi. (Kompajlirana verzija se nalazi u direktorijumu Bin, a njegov izvorni kod u poddirektorijumu ActiveX direktorijuma Demos.)

- Možete da prepustite programu za izradu instalacije da pozove funkciju servera.

Kada ste registrovali server, možete da se okrenete klijent strani našeg primera. Ovoga puta primer je nazvan TestCOM i čuva se u zasebnom direktorijumu. Zapravo, program učitava server DLL preko OLE/COM mehanizma, zahvaljujući informacijama o serveru koje se nalaze u Registryju, tako da klijent ne mora da zna u kom direktorijumu se nalazi server.

Formular koji prikazuje ovaj program je veoma sličan formularu koji smo koristili za testiranje objekta koji se nalazi unutar DLL-a. U klijent programu morate uključiti fajl sa izvornim kodom interfejsa i ponovo deklarisati GUID COM servera. Naravno, kod metoda `FormCreate` ovog programa se mora ažurirati da bi se kreirali potrebni COM objekti. Na početku izvršavanja programa sve kontrole su nedostupne (u vreme dizajniranja), a postaju dostupne tek kada se objekat kreira. Na ovaj način, ukoliko se pozove izuzetak prilikom kreiranja jednog od objekata, kontrole koje se odnose na objekat neće biti dostupne:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  // create first object
  Num1 := CreateComObject (Cass.Number) as Number;
  Num1.SetValue (SpinEdit1.Value);
  Label1.Caption := 'Numi1: ' + IntToStr (Num1.GetValue);
  Button1.Enabled := True;
  Button2.Enabled := True;

  // create second object
  Num2 := CreateComObject (Class_Number) as INumber;
  Label2.Caption := 'Num2: ' + IntToStr (Num2.GetValue);
  Button3.Enabled := True;
  Button4.Enabled := True;
end;
```

Naročito obratite pažnju na poziv `CreateComObject` i narednu as konverziju. API poziv pokreće COM mehanizam konstrukcije objekta, koji sam već detaljno opisao. Ovaj poziv takođe dinamički učitava server DLL. Povratna vrednost je objekat `IUnknown`. Ovaj objekat mora da se konvertuje u odgovarajući tip interfejsa pre nego što mu se dodele polja `Num1` i `Num2`, koja sada imaju tip interfejsa `INumber` za njihov tip podataka:

```
type
  TForm1 = class(TForm)
    ...
  private
    Num1, Num2 : INumber;
```

UPOZORENJE

Da biste preveli interfejs na stvarni tip, uvek koristite `as` konverziju, koja za interfejse obavlja poziv `QueryInterface`. Ovim se obezbeđuje nekakva zaštita, jer se poziva izuzetak ukoliko dati objekat ne podržava interfejs koji konvertujete. U slučaju interfejsa, `as` konverzija je jedini način za izdvajanje interfejsa iz nekog drugog interfejsa. Ukoliko napišete običnu konverziju za `INumber(CreateComObject (Class_Number))`, program će "pući", čak i kada izgleda da konverzija ima smisla kao u prethodnom slučaju. Konvertovanje pokazivača interfejsa u drugi pokazivač interfejsa je greška. Greška i tačka. Nikada to nemojte činiti. ■

Na slici 15.4 možete videti izlaz programa za testiranje, koji je veoma sličan prethodnoj verziji. Primetićete da ovoga puta `Num2` prikazuje početnu vrednost objekta prilikom pokretanja, što predstavlja podešavanje metoda `Initialize`. Takođe ćete primetiti da sam dodao još jednu kontrolu, kojom se kreira treći privremeni COM objekat:

```
procedure TForm1.Button5Click (Sender: TObject);
var
  Num3: INumber;
begin
  // create a new temporary COM object
  Num3 := CreateComObject (Class.Number) as INumber;
  Num3.SetValue (100);
  Num3.Increase;
  ShowMessage ('Num3: ' + IntToStr (Num3.GetValue));
end;
```

SLIKA 15.4 *Izlaz primera TestCom, COM klijent*

Kada kliknete ovu kontrolu, dobijate vrednost broja iza 100. Da biste videli zašto sam dodao ovaj metod primeru, potrebno je da još jednom kliknete kontrolu, kada se prikaže poruka koja prikazuje rezultat. Sada ćete dobiti drugu poruku koja Vas obaveštava da je objekat uklonjen. Ovim se pokazuje da kada se dopusti da interfejs objekat izađe iz opsega, automatski se poziva metod `Release` objekta, smanjuje broj referenci na objekat, i objekat se uklanja ukoliko je broj referenci na objekat jednak nuli. U Poglavlju 3 je ovaj mehanizam prebrojavanja referenci detaljnije opisan.

Isto se dešava i za druga dva objekta čim se prekine izvršavanje programa. Čak i kada program ne ukloni eksplicitno dva objekta u metodu `FormDestroy`, oni se zaista uklanjaju, kao što to jasno ukazuje poruka njihovih `Destroy` destruktora. Ovo se dešava jer su deklarisani kao interfejs tip, i Delfi će za njih koristiti prebrojavanje referenci.

Upotreba svojstava interfejsa

Kao mali dodatak, naš primer možemo proširiti dodavanjem svojstva `INumber` interfejsu. Kada interfejsu dodate svojstvo, Vi naznačavate tip podataka, a tek zatim direktive `read` i `write`. Morate da imate svojstva samo za čitanje i svojstva samo za pisanje, ali klauzule `read` i `write` uvek moraju da se referišu na metod jer interfejsi ne sadrže ništa drugo osim metoda.

Evo ažuriranog interfejsa koji je deo primera PropCom:

```
type
  INumberProp = interface
    [' {B35C5800-8E59-98D0-444553540000}' ]
    function GetValue: Integer; stdcall;
    procedure SetValue (New: Integer); stdcall;
    property Value: Integer
      read GetValue write SetValue;
    procedure Increase; stdcall;
  end;
```

Ja sam ovom interfejsu dodelio novi naziv i, što je mnogo važnije, novi ID interfejsa. Mogao sam da novi tip interfejsa izvedem iz prethodnog, ali time ne bih ništa dobio. COM sam po sebi zaista ne podržava nasleđivanje i, iz perspektive COM-a, svi interfejsi su različiti jednostavno zato što imaju različite ID-ove. Nepotrebno je istaći da u Delphiju nasleđivanje koristimo za poboljšanje strukture koda interfejsa i server objekata koji ih koriste.

U primeru PropCom ja sam ažurirao deklaraciju server klase tako što sam napisao:

```
type
  TDllNumber = class (TComObject, INumberProp)
  ...
```

Ova klasa takođe sadrži novi ID server objekta. Klijent program, koji se nalazi u direktorijumu TestProp, sada jednostavno može da koristi svojstvo `Value` umesto metoda `SetValue` i `GetValue`. Evo malog dela koda metoda `FormCreate`:

```
Num1 := CreateComObject (Class_NumPropServer) as INumberProp;
Num1.Value := SpinEdit1.Value;
Label1.Caption := 'Num1: ' + IntToStr (Num1.Value);
```

Razlika između upotrebe metoda i svojstava za interfejse je samo sintaksička, jer svojstva interfejsa ne mogu da pristupaju privatnim podacima kao što mogu svojstva klase. Upotrebom svojstava kod možemo učiniti čitljivijim.

Pozivanje virtuelnih metoda

Izradili smo nekoliko primera zasnovanih na COM-u, ali se možda još uvek nelagodno osećate kada pomislite na program koji poziva metode objekata koji su kreirani unutar DLL-a. Kako je ovo moguće ukoliko ove metode nije izvezao DLL? COM server, dakle DLL, kreira objekat i predaje ga aplikaciji koja vrši poziv. Kada to učini, DLL kreira objekat upotrebom tabele virtuelnih metoda. (Ne zaboravite da su svi interfejs metodi virtuelni po definiciji.)

Kako svaki objekat ugnežđuje pokazivač na svoju tabelu virtuelnih metoda, glavni program prihvata objekat, a takođe i način na koji će sa njim raditi pozivanjem njegovih virtuelnih metoda. Glavni program ne mora da zna memorijske adrese ovih metoda, jer ih zna objekat, baš kao što je slučaj sa polimorfnim pozivom. Međutim, COM je još moćniji od ovog: nije potrebno da znate koji programski jezik je upotrebljen za kreiranje objekta, jer njegov VMT poštuje standard koji uspostavlja COM.

SAVET

COM kompatibilni VMT takođe ima i čudan efekat. Nazivi metoda nisu bitni kada se obezbede njihove adrese na pravim mestima u VMT-u. To je razlog što možete mapirati metod interfejsa za aktuelnu funkciju koju implementirate. ■

Kao rezime možemo reći da COM obezbeđuje binarni standard za objekte koji ne zavisi od programskih jezika. Objekti koje delite između modela su kompajlirani, a njihovi VMT-ovi imaju određenu strukturu, koju definiše COM a ne razvojno okruženje koje ste koristili.

Upotreba interfejs školjke

U poslednjem odeljku smo izradili standardni COM objekat, koji smo spakovali u server u procesu, a koristili smo ga sa standardnog klijenta. Ipak, COM interfejs koji smo implementirali bio je uobičajeni interfejs. Sada možemo pokušati da izradimo klijente i servere koji se odnose na Windows interfejs školjku, koja se u potpunosti zasniva na COM-u. Originalni Windows API je u osnovi kolekcija funkcija, ali svi najnoviji API-ji su zasnovani na COM-u.

Naredni odeljci koriste postojeće servere koji su deo Windows školjke; u ovom slučaju ćemo napisati klijent aplikaciju i koristićemo COM servere koje obezbeđuje sistem. Ovaj slučaj ilustruje razliku u odnosu na tradicionalnu upotrebu Windows API poziva. Takođe ću napisati neke COM servere koje će koristiti Windows sistem, naročito Explorer. Ovaj slučaj ilustruje razliku u odnosu na tradicionalno programiranje funkcije koju poziva sistem.

Kreiranje prečica

Jedan od najjednostavnijih interfejsa školjki koji možemo da koristimo u klijent aplikaciji je interfejs `IShellLink`. Ovaj interfejs povezuje Windows sa prečicama i omogućava programerima da pristupe informacijama postojećih prečica ili da kreiraju nove prečice. U našem primeru ShCut ja ću kreirati različite vrste prečica, ali će se sve odnositi na sam program. Naravno, kada jednom ovo budete shvatili, moći ćete lako da proširite primer i da kreirate prečice za bilo koji program ili fajl.

Primer sadrži polje za izmene koje se koristi za naziv prečice, nekoliko polja za potvrdu i dve kontrole. Kada se klikne kontrola Create, tekst polja za izmene se koristi za naziv nove prečice, koja se smešta u trenutni direktorijum, na radnu površinu ili u meni Start. Ove opcije nisu ekskluzivne; korisnik može odjednom da kreira više prečica.

Najvažniji kod je na samom početku ovog metoda. Poziv `CreateComObject` kreira sistemski objekat, što je naznačeno GUID-om koji se prosleđuje kao parametar. Rezultat ovog poziva (koji je interfejs `IUnknown`) se konvertuje u interfejs `IShellLink` i interfejs `IPersistFile`:

```
uses
  ComObj, ActiveX, ShlObj, Registry;

procedure TForm1.ButtonlClick(Sender: TObject);
var
  AnObj; IUnknown;
  ShLink: IShellLink;
  PFile: IPersistFile;
  FileName: string;
  WFileName: WideString;
```

```
  Reg: TRegIniFile;
begin
  // access the two interfaces of the object
  AnObj := CreateComObject (CLSID_ShellLink)'
  ShLink   := AnObj as tShellLink;
  PFile := AnObj as IPersistFile;
```

Zapravo, mogli smo da napišemo tri prethodne linije koda koristeći kraću notaciju:

```
ShLink := CreateComObject (CLSID_ShellLink) as IShellLink;
PFile := ShLink as IPersistFile;
```

Ukoliko pogledate slične primere izrađene u drugim programskim jezicima, primetićete da program koristi uobičajene pozive metoda `QueryInterface` za pristup interfejsu `IPersistFile`. Dva as izraza u osnovi pozivaju `QueryInterface` za nas.

Kada dobijemo `IShellInterface`, možemo pozivati neke od njegovih metoda, kao što su `SetPAth` i `SetWorkingDirectory`:

```
// get the name of the application file
FileName := ParamStr (0);
// set the link properties
ShLink.SetPath (PChar (FileName));
ShLink.SetWorkingdirectory (PChar (
ExtractFilePath (FileName)));
```

Kada jednom podesimo objekat veze školjke, moramo ga sačuvati, već prema statusu tri polja za potvrdu, pozivajući metod `Save` interfejsa `IPersistFile` objekta. Najjednostavnija verzija je ona koja se koristi za čuvanje veze u trenutnom direktorijumu:

```
// save the file in the current dir
if cbDir.Checked then
begin
  // using a WideString
  WFileName := ExtractFilePath (FileName) +
    EditName.Text + '.lnk'
  PFile.Save (PWChar (WFileName), False);
end;
```

Poziv metoda Save (koji kreira fizički LINK fajl) zahteva parametar "pointer to wide char". Najjednostavniji način na koji ovo možemo da postignemo je da deklarišemo string i da ga zatim konvertujemo u `PWChar`. Nemojte pokušavati da običan string konvertujete u `PWChar` — kompajler se neće buniti, ali program neće funkcionisati!

Da biste kreirali prečicu na radnoj površini ili u meniju Start, potrebno je da prvo odredimo odgovarajući sistemski direktorijum pronalaženjem odgovarajuće vrednosti u Registryju. Pisanjem programa na ovaj način osiguravamo da će raditi na različitim verzijama Windowsa kao i na lokalizovanim verzijama. Evo izvornog koda poslednja dva polja za potvrdu:

```
// save on the desktop
if cbDesktop.Checked then
begin
  Req := TReqIniFile.Create (
    'Software\MicroSoft\Windows\CurrentVersion\Explorer'');
  WFileName := Reg.ReadString ('Shell Folders', 'Desktop', ' ') +
```

```
    '\' +  EditName.Text + ' .lnk';
  Req. Free;
  PFile.Save (PWChar (WFileName), False);
end;
// save in the Start Menu
if cbStartMenu.Checked then
begin
  Req := TReqIniFile.Create (
    'Software\MicroSoft\Windows\CurrentVersion\Explorer'');
  WFileName := Reg.ReadString ('Shell Folders', 'Start Menu', ' ') +
    '\' +  EditName.Text + ' .lnk';
  Req. Free;
  PFile.Save (PWChar (WFileName), False);
end;
```

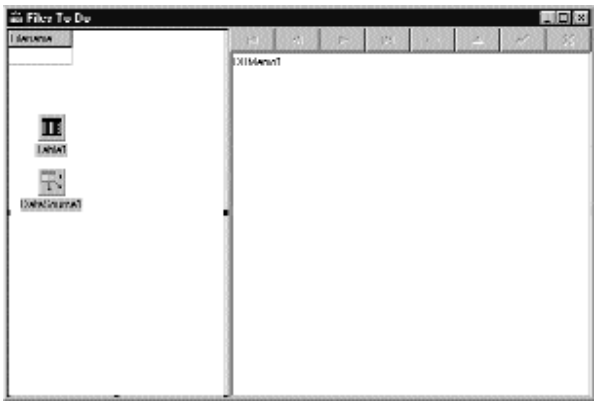
Da bih pronašao informaciju u Registryju, ja sam upotrebio klasu `TRegIniFile`, mada postoje i druge klase VCL-a koje se mogu upotrebiti, kao što je klasa `TRegistry`. Efekat izvršavanja ovog programa i efekat kada kliknete kontrolu je da će Windows dodati novi link u direktorijum projekta, na radnu površinu ili meni Start. Primer programa možete videti na slici 15.5.

SLIKA 15.5 *Jednostavan korisnički interfejs primera ShCut i dve prečice koje su kreirane pomoću programa u direktorijumu projekta i na radnoj površini*

Aplikacija To-Do-File

Za drugi primer integracije Delphi programa sa sistemskom školjkom ja sam pokušao da napišem jednostavnu praktičnu aplikaciju koja koristi prevlačenje fajlova i obradu kontekst menija. Prvo ću početi sa prevlačenjem fajlova, jer će to predstaviti neke tehnike koje koristi obrada kontekst menija.

Kao što sam pomenuo, ova aplikacija je zaista korisna; možete je koristiti za kreiranje to-do-liste. Zasniva se na tabeli Paradox koja čuva nazive fajlova i beleške o fajlovima. Formular aplikacije sadrži komponentu DBGrid, koja prikazuje samo jednu kolonu koja sadrži nazive fajlova, i kontrolu Memo, koja sadrži beleške o aktuelnom fajlu. Formular u vreme dizajniranja možete videti na slici 15.6.



SLIKA 15.6 *Formular primera ToDoFile u vreme dizajniranja*

SAVET

Upotreba DBGrida sa jednom kolonom je jedini način da u Delphiju prikažete spisak raspoloživih slogova u formatu Listbox. Alternativa je, naravno, da popunite ListBox koristeći kod i da se zatim ručno krećete kroz tabelu baze podataka kada se selekcija u ListBoxu promeni. Ovaj manuelni pristup je manje efikasan kada imamo veliki broj slogova jer program treba da ih sve pretraži da bi popunio ListBox, dok DBGrid učitava samo slog koji trenutno prikazuje. ■

Primetićete da komponenta Navigator ne sadrži kontrolu za novi slog i da je komponenta DBGrid podešena samo za čitanje. Zapravo, korisnik ne bi trebalo da ima mogućnost da kreira nove slogove izuzev prevlačenjem fajlova na formular, i nije mu omogućeno da promeni polje sa nazivom fajla na bilo koji način (izuzev da ga ukloni). Sve što korisnik može da učini jeste da izmeni polje za beleške, to jest, da unese opis operacija koje treba da se izvrše nad fajlom.

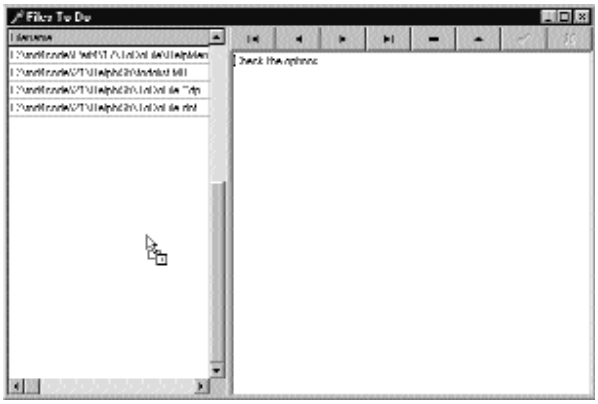
Kreiranje baze podataka

Da bih kreirao tabelu baze podataka za ovaj primer, ja sam koristio svojstvo `FieldDefs` da bih definisao strukturu, i odredio vrednost `True` za svojstvo `StoreDefs` da bih sačuvao definiciju tabele uz DFM fajl formulara. Tabela sadrži dva polja, polje za string nazvano Filename i Memo polje nazvano Notes. Naravno, tabelu takođe možete kreirati i u vreme izvršavanja upotrebom lokalnog menija komponente Table. Program, ipak, poziva metod `CreateTable` u obradi događaja `OnCreate`, izuzev ukoliko to već nije urađeno:

```
procedure TToDoFileForm.FormCreate (Sender: Tobject);
begin
  // eventually create the table
  if not Table1.Exists then
    Table1.CreateTable;
  // activate the table
  Table1.Activate;
  // accept dragging to the form
  DragAcceptFiles (Handle, True);
end;
```

Prevlačenje fajlova na formular

Kao što možete videti u prethodnom listingu, kod inicijalizacije formulara takođe registruje prozor u sistemu kao odredište za prevlačenje fajlova, pozivajući Windows API funkciju `DragAcceptFiles`. Kao rezultat, kursor aplikacije se pretvara u tipičnu ikonu "drag accept" (prihvata prevlačenje) kada se fajl prevuče na formular. Primer ovog kursora možete videti na slici 15.7.



SLIKA 15.7 *Kursor "prihvata prevlačenje" koji prikazuje aplikacija ToDoFile kada korisnik prevuče fajl iznad formulara*

Kada se obavi operacija prevlačenja fajla, sistem prozoru šalje poruku `wm_DropFiles`. Ova poruka prosleđuje (između ostalih parametara) hendl na strukturu file-drop (predavanja fajla) iz koje možete izdvojiti informaciju upotrebom API funkcije `DragQueryFile`. Kada se ova API funkcija pozove parametrom `$FFFFFFFF`, kao rezultat daje broj fajlova koji se prevlače na prozor; kada se pozove upotrebom numeričkog parametra, popunjava bafer nazivom fajla. Zbog ovoga kod obrade poruke `wm_DropFiles` prvo dobija broj fajlova, a zatim prolazi kroz petlju za svaki od fajlova, kao što to pokazuje naredni listing:

```
procedure TToDoFileForm.DropFiles(var Msg: TWmDropFiles);
var
  nFiles, I: Integer;
  Filename: string;
begin
  // get the number of dropped files
  nFiles := DragQueryFile (Msg.Drop, $FFFFFFFF, nil, 0);
  // for each file
  try
    for I := 0 to nFiles - 1 do
    begin
      // allocate memory
      SetLength (Filename, 80);
      // read the file name
      DragQueryFile (Msg.Drop, I, PChar (Filename), 80);
      // normalize file
      Filename := PChar (Filename);
      // add a new record
```

```
      Table1.InsertRecord ([Filename, ' ']);
    end;
  finally
    DragFinish (Msg Drop);
  end;
  // open the (last) record in edit mode
  Table1. Edit;
  // move the input focus to the memo
  DBMemo1.SetFocus;
end;
```

Kao što možete videti u prethodnom kodu, za svaki novi fajl program umeće novi slog, sa odgovarajućim nazivom fajla i praznim poljem za beleške. Zatim, za poslednji fajl koji se prevlači, program otvara slog u modu za izmene i pomera fokus na kontrolu Memo, tako da korisnik može da unese beleške za taj fajl.

Kreiranje obrade kontekst menija

Sada kada imamo osnovni program koji se izvršava, sistemu možemo da dodamo proširenje školjke, da bismo korisniku omogućili da jednostavno selektuje fajl i da ga "pošalje" aplikaciji, a da ne mora da obavi operaciju prevlačenja, koja nije uvek zgodna kada se istovremeno izvršava više programa. Proširenje kontekst menija je jedno od raspoloživih Windows proširenja školjke i aktivira se svaki put kada korisnik klikne desnim tasterom miša fajl u Windows Exploreru.

Kontekst meni je COM server koji pokazuje interni objekat koji će se kreirati i koristiti od strane sistema. Kontekst meni COM objekat mora da implementira dva interfejsa, interfejse `IContextMenu` i `IShellExtInit`. Prvi interfejs definiše specifične akcije za kontekst meni, kao što je definisanje broja elemenata menija koji se dodaju i njihov tekst, dok drugi interfejs definiše način za pristupanje fajlu ili fajlovima kojima korisnik operiše. Sledi rezultujuća definicija COM server klase objekta:

```
type
  TToDoMenu = class(TComObject, IUnknown,
    IContextMenu, IShellExtInit)
  private
    fFileName: string;
  protected
    {Declare IContextMenu methods here}
    function QueryContextMenu(Menu: HMENU; indexMenu,
      idCmdFirst, idCmdLast, uFlags: WENT): HResult; stdcall;
    function InvokeCommand(
      var lpici: TCMlnvokeCommandInfo): HResult; stdcall;
    function GetCommandString(idCmd, uType: UINT;
      pwReserved: PUINT; pszName: LPSTR;
      cchMax: WENT): HResult; stdcall;
    {Declare IShellExtlnir methods here}
    function IShellExtInit.Initlalize.InitShellExt;
    function InitShellExt (pidlFolder: PItemIDList;
      lpdobj: ]IDataObject; hKeyProgID: HKEY): HResult; stdcall;
end;
```

Primetićete da klasa implementira metod Initialize interfejsa `IShellExtInit` ali sa drugačije imenovanim metodom, `InitShellExt`. Razlog za ovo je što sam hteo da izbegnem da se ovaj metod pomeša sa metodom `Initialize` osnovne klase `TComObject`, koja predstavlja vezu koju treba da inicijalizujemo za objekat, kao što je opisano ranije u ovom poglavlju. Proučimo metod prvo `InitShellExt`; on je svakako najkomplikovaniji od svih metoda:

```
function TToDoMenu.InitShellExt(pidlFolder: PItemIDList;
  lpdobj: IDataObject; hkeyProglED: HKEY): RResult; stdcall;
var
  medium: TStgMedium;
  fe: FormatEtc;
begin
  Result := E_FAIL;
  // check if the lpdobj pointer is nil
  if Assigned (lpdobj) then
  begin
    with fe do
    begin
      cfFormat := CF_HDROP;
      ptd := nil;
      dwAspect := DVASPEC_CONTENT;
      lindex := -1;
      tymed := TYMED_HGLOBAL;
    end;
    // transform the lpdobj data to a storage medium structure
    Result := lpdobj.GetData(fe, medium);
    if not Failed (Result) then
    begin
      // check if only one file is selected
      if DragQueryFile (medium.hGlobal, $FFFFFFFF, nil, 0) 1 then
      begin
        SetLength (fFileName, 1000);
        DragQueryFile (medium.hGlobal, 0, PChar (fFileName), 1000);
        // realign string
        fFileName := PChar (fFileName);
        Result := NOERROR;
      end
      else
        Result := E_FAIL;
    end;
    ReleaseStgMedium(medium);
  end;
end;
```

Inicijalni deo metoda transformiše pokazivač na interfejs `IDataObject`, koji dobijamo kao parametar, u istu strukturu podataka koju smo koristili za operaciju prevlačenja fajla, tako da možemo da pročitamo informaciju o fajlu upotrebom funkcije `DragQueryFile`. Komplikovani način kodiranja je, zapravo, najjednostavniji koji možete da upotrebite! Na kraju ove operacije dobijamo vrednost naziva fajla. Bilo kakva selekcija više fajlova se ne prihvata.

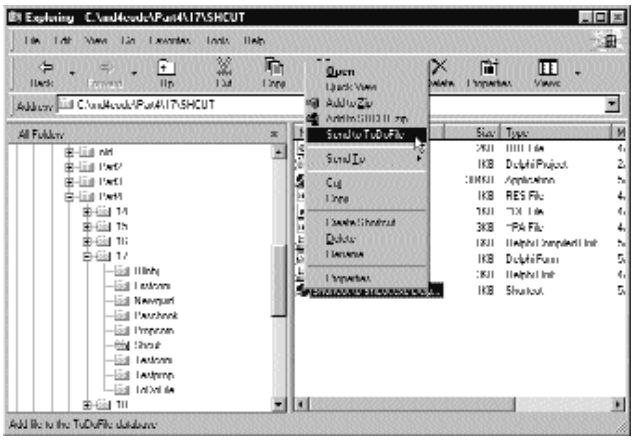
Sada možemo da proučimo metode interfejsa `IContextMenu`. Prvo, `QueryContextMenu` se koristi za dodavanje novih elemenata lokalnom meniju fajla. U ovom slučaju, dodajemo novi element menija (pozivanjem API funkcije `InsertMenu`), ali samo ukoliko se aplikacija ToDoFile izvršava.

Da bismo utvrdili da li se aplikacija izvršava, možemo da potražimo prozor koji odgovara klasi `TToDoFileForm`, koji bi trebalo da bude jedinstven u sistemu. Rezultat funkcije je broj elemenata koji su dodati meniju:

```
function TToDoMenu.QueryContextMenu(Menu: HMENU;
  indexMenu, idCmdFirst, idCmdLast, uFlags: UINT): HResult;
begin
  // add entry only if the program is running
  if FindWindow ('TToDoFileForm' , nil) K> 0 then
  begin
    // add a new item to context menu
    InsertMenu (Menu, indexMenu,
      MF_STRING or MF_BYPOSITI0N, idCmdFirst,
      'Send to ToDoFile');
    // return the number of menu items added
    Result := 1;
  end
  else
    Result := 0;
end;
```

Sada kada su elementi dodati meniju, korisnik može da ih odabere. Dok on ili ona prelaze preko elemenata, prikazuje se opis na statusnoj liniji Windows Explorera, kao što možete videti na slici 15.8. Meni ID (`idCmd`) koji dobijamo u metodu `GetCommandString` je samo relativni broj, koji počinje od nule, elemenata koje smo dodali meniju. Kada se kursor nalazi iznad elementa, mi samo kopiramo string sa njegovim opisom u bafer koji obezbeđuje sistem:

```
function TToDoMenu.GetCommandString(idCmd, uType: UINT;
  pwReserved: PUINT; pszName: LPSTR; cchMax: UINT): HRESULT;
begin
  if idCmd = 0 then
  begin
    // return help string for menu item
    strCopy (pszName, 'Add file to the ToDoFile database');
    Result := NOERROR;
  end
  else
    Result := E_INVALIDARG;
end;
```



SLIKA 15.8 *Kada korisnik pomeri pokazivač miša iznad novog elementa lokalnog menija u Windows Exploreru, prikazuje se opis na statusnoj liniji*

Poslednji korak je operacija koju treba obaviti kada se element menija zaista odabere. Metod `InvokeCommand` dobija pokazivač na strukturu koja čuva zahtev. Ovaj metod poštuje standardnu šemu da prvo proveri da li je zahtev validan, proverom dve 16-bitne reči vrednosti `lpici.lpVerb`. Posle ovih preliminarnih (ali obaveznih) koraka, proveravamo vrednost da bismo videli koji element menija je aktiviran; ili, ukoliko kontekst meni sadrži samo jedan element, kao u ovom slučaju, mi samo proveravamo da li je vrednost nula. Sledi struktura koda, pre nego što dodamo specifičnu akciju:

```
function TToDoMenu.InvokeCommand (
  var lpici: TCMInvokeCommandInfo): HResult;
begin
  Result := NOERROR;
  // make sure we are not being called by an application
  if HiWord(Integer(lpici.lpverb)) <> 0 then
  begin
    Result := E_FAIL;
    Exit;
  end;
  // make sure we aren't being passed an invalid argument number
  if LoWord(lpici.lpVerb) > 0 then
  begin
    Result := E_INVALIDARG;
    Exit;
  end;
  // execute the command specified by lpici.lpVerb
  if LoWord(lpici.lpVerb) = 0 then
  begin
    // actual code still missing here
  end
end;
```

Slanje podataka drugoj aplikaciji upotrebom poruke wm_CopyData

Pošto imamo naziv fajla sa kojim korisnik radi, sve što je potrebno da uradimo u obradi kontekst menija jeste da pošaljemo taj naziv glavnom formularu aplikacije ToDoFile. Problem je što se DLL obrada kontekst menija izvršava u procesu Windows Explorera, te ne može da pošalje vrednost memorijskog pokazivača drugom procesu. To bi jednostavno bilo beskorisno, jer pod Windowsom 32 različite aplikacije imaju zasebne memorijske adresne prostore.

U prethodnom poglavlju smo videli da je jedan način za deljenje podataka između aplikacija upotreba memorijski mapiranog fajla. Druga tehnika, koja je u ovom slučaju bolja, jeste upotreba poruke `wm_CopyData`. To je specijalna Windows poruka koja može da se koristi za slanje memorijskog bafera drugoj aplikaciji: Windows će za nas rešiti sve probleme oko konverzije memorije. Program u osnovi popunjava podacima strukturu podataka `CopyDataStruct` i naznačava njihovu dužinu, a zatim mora da upotrebi API funkciju `SendMessage` da bi je prosledio odredišnom prozoru. Iz ovog razloga moramo ponovo da upotrebimo `FindWindow` da bismo dobili hendl glavnog prozora aplikacije ToDoFile. Evo ostatka koda metoda `InvokeCommand`:

```
var
  hwnd: THandle;
  cds: CopyDataStruct;
begin
  ...
  if LoWord(lpici.lpVerb) = 0 then
  begin
    // get the handle of the window
    hwnd := FindWindow ('TToDoFileForm', nil);
    if hwnd <> 0 then
    begin
      // prepare the data to copy
      cds.dwData := 0;
      cds.cbData := length (fFileName);
      cds.lpData := PChar (fFileName);
      // acrivate the destination window
      SetForegroundWindow (hwnd);
      // send the data
      SendMessage (hwnd, wm_CopyData,
        lpici.hwnd, Integer (@cds));
  end;
end;
```

NAPOMENA

Pre slanja podataka moramo da aktiviramo odredišni prozor pozivanjem API funkcije `SetForegroundWindow`. Ovo je neophodno jer se spremamo da aktiviramo prozor koji je kreiran nekim drugim procesom, nešto što Windows obično ne čini. Takođe ćete primetiti da ukoliko ovaj poziv napišete u aplikaciji ToDoFile kada primi poruku `wm_CopyData`, on neće načiniti nikakav efekat. ■

Kada obrada kontekst menija pošalje podatke, aplikacija se mora proširiti da bi obradila poruku `wm_CopyData`. U ovoj obradi događaja mi dobijamo istu strukturu koju smo poslali drugoj strani, mada između operacije slanja koju obavlja obrada kontekst menija i operacije prihvatanja aplikacije. Windows se stara o korektnom mapiranju podataka u drugi adresni prostor. Kao rezultat, izdvajanje naziva fajla je zapravo veoma jednostavno, ali imajte na umu da je to samo posledica činjenice

da Windows obavlja veliki deo posla, a da mi to ne vidimo. Upotreba neke druge obične Windows poruke sem `wm_CopyData` nikada ne bi funkcionisala!

Evo koda koji sam dodao formularu aplikacije ToDoFile. Kod obavlja nekoliko stvari: restaurira aplikaciju ukoliko je minimizirana, izdvaja naziv fajla, umeće novi slog u tabelu baze podataka, kopira naziv fajla i još jednom pomera fokus na Memo kontrolu.

```
procedure TToDoFileForm.CopyData(var Msg: TWmCopyData);
var
  Filename: string;
begin
  // restore the window if minimized
  if IsIconic (Application.Handle) then
    Application. Restore;

  // extract the filename from the data
  Filename := PChar (Msg.CopyDataStruct.lpData);
  // insert a new record
  Table1. Insert;
  // set up the file name
  Table1.FieldByName (Filename').AsString := Filename;
  // move the input focus to the memo
  DBMemo1. SetFocus;
end;
```

Registrovanje proširenja školjke

Posle pisanja ovog proširenja školjke moramo je i registrovati. Upotrebom uobičajene Run➥Register ActiveX komande možemo registrovati server na sistemu, ali ipak moramo da obezbedimo neke dodatne informacije da bismo je registrovali kao proširenje školjke, u našem slučaju za bilo koji tip fajla. Postoji nekoliko pristupa, ne računajući ručno editovanje Registryja. Možete da napišete REG fajl prema sledećoj strukturi:

```
REGEDIT4

[HKEY_CLASSES_ROOT\CLSID\{CDF05220-DB84-11D1-B9F1-004845400FAA}]
@= 'ToDoFile Context Menu'
[HKEY_CLASSES_ROOT\CLSID\ {CDF05220-DB84-11D1-B9F1-004845400FAA}
  \InProcServer32]
@= "c:\\md5code\\Part4\\15\\ToDoFile\\ToDoShll.dll"
'ThreadingModel" = "Apartment"

[HKEY_CLASSES_ROOT\*\shellex\ContextMenuHandlers\
  {CDF05220-DB84-11D1-B9F1-004845400FAA}]
@= " "
```

Prvi deo ovog fajla odgovara registraciji koja je već obavljena instaliranjem servera, dok poslednji deo dodaje server kao obradu kontekst menija za sve fajlove (što je naznačeno simbolom * u putanji `HKEY_CLASSES_ROOT`).

Potpuno drugačiji, ali mnogo bolji pristup je dodavanje informacije o registraciji direktno u COM server biblioteku. Unapred određena registracija se odvija u klasi `TComObjectFactory` kada

se izvrši metod UpdateRegistry. Unapred određenu registraciju možemo da promenimo izvođenjem klase iz standardne klase radionice klase i zaobilaženjem ovog metoda:

```
type
  TToDoMenuFactory = class (TComObjectFactory)
  public
    procedure UpdateRegistry (Register: Boolean); override;
end;
```

U ovom metodu je potrebno ili da dodamo element u Registry ili da ga uklonimo, već prema vrednosti parametra Boolean:

```
procedure TToDoMenuFactory.UpdateRegistry(Register: Boolean);
var
  Reg: TRegistry;
begin
  inherited UpdateRsgistry (Register);

  Req := TRegistry.Create:
    try
      // register or remove the menu handler
      if Register then
        Reg.CreateKey (
          \HKEY_CLASSES_ROOT\*\Shellex\ContextMenuHandler\ ' +
        GUIDToString (Class_ToDoMenuMenu))
    else
      Reg.DeleteKey (
        \HKEY_CLASSES_ROOT\*\Shellex\ContextMenuHandler\ ' +
        GUIDToString (Class_ToDoMenuMenu));
  finally
    Reg.Free;
  end;
end;
```

U inicijalizacionom odeljku jedinice COM objekta takođe je potrebno da kreiramo novi globalni objekat ove klase umesto osnovne klase radionice klase:

```
initialization
  TToDoMenuFactory.Create (
    ComServer, TToDoMenu, Class_ToDoMenuMenu,
    'ToDoMenu', 'ToDoMenu Shell Extension',
    ciMultiInstance, tmApartment);
```

Sada jednostavno možete da registrujete server i podesite ga prema obradi kontekst menija upotrebom Delphijeve Run➥Register ActiveX komande menija, aplikacije RegSrv32, ili većine alata koji se koriste za kreiranje instalacionih programa.

Šta je sledeće?

U ovom poglavlju sam razmatrao osnove Microsoftove COM tehnologije. Videli smo kako Delphi podržava COM i kako Delphi čini programiranje Explorer proširenja veoma lakim.

Naredno poglavlje nas vodi do COM tehnika višeg nivoa obrađujući Automation, Documents i Controls. Sada kada znamo osnove, istraživanje ovh COM tehnologija će svakako biti lakše.

Na ostale elemente COM-a ćemo se vratiti kada budemo razmatrali Internet aplikacije i distribuirane aplikacije u Poglavlju 20.

POGLAVLJE

16

Automatizacija i ActiveX

Posle poslednjeg poglavlja, koje je bilo posvećeno osnovama Microsoftove COM arhitekture, vreme je da proučimo neke Windows tehnike programiranja visokog nivoa koje su zasnovane na COM-u. Počećemo razmatranjem Automationa i uloge editora Type Libraries. Takođe, videćmo kako da pravilno radimo sa Delphi tipovima podataka u Automation serverima i klijentima.

Kasnije ćemo obratiti pažnju na upotrebu Automation podrške koju obezbeđuju Microsoft Office aplikacije i koju Delphi 5 čini još jednostavnijom zahvaljujući nekim komponentama koje su spremne za upotrebu, a koje su smeštene u Office server programe i dokumente.

U poslednjem delu poglavlja istražićemo upotrebu ugnežđenih objekata upotrebom komponente OleContainer i istražićemo programiranje OLE kontrola ili ActiveX kontrola. Međutim, počnimo od osnovnog materijala.

OLE Automation

U prethodnom poglavlju smo videli da možete da koristite COM da biste omogućili da izvršni fajl i biblioteka dele objekte i da se ovaj postupak može koristiti za interakciju sa Windows školjkom. U većem broju slučajeva, korisnici ipak žele aplikacije koje mogu međusobno da komuniciraju. Jedan od pristupa koji možete koristiti da biste ostvarili ovakav zahtev je OLE Automation. Posle nekoliko primera koji upotrebljavaju korisničke interfejse na osnovu biblioteka tipa, ja ću objasniti programiranje Word i Excel OLE kontrolera, pokazujući kako da izvršite transfer informacije baze podataka u te aplikacije.

NAPOMENA

U trenutnoj Microsoftovoj dokumentaciji koristi se termin Automation umesto termina OLE Automation, i koriste se termini aktivni dokument (active document) i složeni dokument (compound document) umesto termina OLE Document. U ovoj knjizi se koristi nova terminologija pored starije "OLE" terminologije koja je ugrađena u mnoge nazive Delphi komponenata i druge identifikatore. ■

U Windowsu aplikacije ne žive u odvojenim svetovima; korisnici često žele da imaju interakciju između aplikacija. Clipboard i DDE nude veoma jednostavan način za interakciju aplikacija, jer korisnici mogu da kopiraju i prebacuju podatke između aplikacija. Ipak, sve više programa nudi OLE Automation interfejs da bi drugim programima omogućili da upravljaju njima. Pored ove očigledne prednosti programirane automatizacije u odnosu na manuelne operacije, ovi interfejsi su potpuno nezavisni od programskog jezika, te možete da koristite Delphi, C++, Visual Basic ili makro jezik da biste upravljali OLE Automation serverom, bez obzira na programski jezik koji je upotrebljen da bi se server napisao.

OLE Automation je veoma jednostavno primeniti u Delphiju, zahvaljujući obimnom poslu koji obavljaju VCL i kompajler da bi korisniku olakšali programiranje. Da bi podržao OLE Automation, Delphi obezbeđuje jednostavan čarobnjak i moćni editor Type Library, a podržava i dualne interfejse.

Kada koristite DLL u procesu, klijent aplikacija može jednostavno da upotrebi server i može direktno da pozove njegove metode jer se nalaze u istom adresnom prostoru. Kada koristite OLE Automation, situacija je mnogo komplikovanija. Klijent (koji se naziva kontroler — controller) i server su dve zasebne aplikacije koje se izvršavaju u zasebnim adresnim prostorima. Zbog toga sistem mora da prosledi pozive metoda putem složenog mehanizma koji se naziva uređivanje (marshalling) — nešto što ja neću detaljno objasniti. Ono što je važno da se zna je da postoje dva načina na koje kontroler može da pozove metode koji su istaknuti serverom:

- Kontroler jednostavno može da zatraži izvršavanje metoda, prosleđujući ime metoda u stringu, na način koji je sličan dinamičkom pozivu DLL-a. To je ono što Delphi čini kada koristite promenljivu prilikom poziva OLE Automation servera. Ovu tehniku je veoma lako koristiti, ali je ona prilično spora i obezbeđuje veoma malo provere tipa u kompajleru.

- Kontroler može da uveze definiciju Delphi interfejsa za svaki objekat servera i poziva njegove metode na direktniji način (jednostavnim slanjem broja). Ova tehnika, koja se zasniva na interfejsima, omogućava kompajleru da proveri tipove parametara i da proizvede brži kod, ali od programera zahteva nešto više napora. Takođe, može se desiti da svoju kontroler aplikaciju povezujete sa specifičnom verzijom servera. Varijacija ove tehnike uključuje upotrebu interfejsa za prosleđivanje, koji se zasniva na definiciji interfejsa.

U narednim primerima ćemo koristiti sve ove tehnike i poredićemo ih među sobom.

Uvod u Type Libraries (biblioteke tipa)

Najvažnija razlika između dva pristupa je da drugi pristup generalno zahteva Type Library, što je jedna od osnova OLE-a i COM-a. Type Library je u osnovi kolekcija informacija o tipu. Ova kolekcija obično opisuje sve elemente (objekte, interfejse i druge informacije o tipu) koje je server učinio dostupnim. Ključna razlika između Type Library i drugih opisa ovih elemenata (kao što je C i Pascal kod) je u tome da je Type Library nezavisan od programskog jezika. OLE definiše elemente tipa kao podskup standardnih elemenata programskog jezika, a može ih koristiti bilo koji programerski alat. Zbog čega nam je potrebna ovakva informacija?

Kao što sam ranije pomenuo, jednostavan OLE Automation kontroler može da koristi promenljive i nema nikakve informacije o tipu servera koji koristi. To znači da, u pozadini, svaki poziv funkcije treba da se prosledi serveru upotrebom metoda `Invoke` interfejsa `IDispatch`, prosleđujući naziv funkcije kao parametar, i nadajući se da naziv funkcije odgovara postojećoj funkciji servera.

Mada ovo izgleda komplikovano, mali fragment koda koji koristi stari Automation interfejs Microsoft Worda, koji je registrovan kao `Word.Basic`, ilustruje koliko je to jednostavno za programera:

```
var
  VarW: Variant;
begin
  VarW := CreateOleObject ('Word.Basic');
  VarW.FileNew;
  VarW.Insert ('Mastering Delphi by Marco Cantu');
```

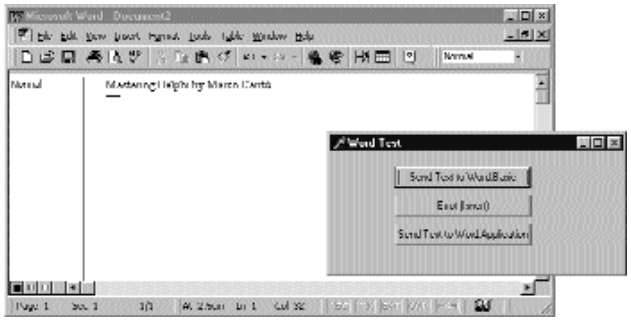
NAPOMENA

Kao što ćemo kasnije videti, Word 97 još uvek registruje Word.Basic interfejs, koji odgovara internom makro jeziku WordBasic, ali registruje i novi Word.Application interfejs koji odgovara makro jeziku VBA. Takođe ćemo videti da Delphi 5 obezbeđuje neke komponente koje pojednostavljuju povezivanje sa aplikacijama Microsoft Officea. ■

Ove tri linije koda pokreću Word (izuzev ukoliko se on već ne izvršava), kreiraju novi dokument i u dokument dodaju nekoliko reči. Efekat ovog koda možete videti na slici 16.1. Promenljiva je tipa podataka type-variant. Može pretpostaviti da su njene vrednosti različitog tipa podataka, uključujući COM objekat koji podržava interfejs IDispatch. Promenljive tipa variant se proveravaju po tipu u vreme izvršavanja; to je razlog zbog kojeg kompajler može da kompajlira kod čak i kada ne zna ništa o metodima OLE Automation servera.

Na nesreću, Delphi kompajler nema način da proveri da li metodi postoje. Proveravanje svih tipova u vreme izvršavanja je rizično, jer ukoliko načinite i najmanju grešku u nazivu funkcije, nećete dobiti nikakvo upozorenje o grešci sve dok ne pokrenete program i dok se prilikom izvršavanja koda ne stigne do linije u kojoj je greška. Na primer, ukoliko otkucate `VarW.Isnert`, kompajler se neće buniti zbog pogrešno unetog naziva, ali će se u vreme izvršavanja prijaviti greška. Kako ne prepoznaje naziv, Word pretpostavlja da metod ne postoji.

Mada OLE `IDispatch` interfejs podržava pristup koji smo videli, takođe je moguće — i bezbednije — za server da izveze opis svojih interfejsa i objekata upotrebljavajući Type Library. Editor Type Library se zatim može konvertovati specifičnim alatima (kao što je Delphi) u definicije napisane u jeziku koji želite da koristite za pisanje klijent ili kontroler programa (kao što je Object Pascal). Ovo omogućava kompajleru da proveri da li je kod korektan.



SLIKA 16.1 *Wordov dokument se kreira i uređuje Delphi aplikacijom WordTest*

Kada kompajler završi proveru, može da koristi dve različite tehnike za slanje zahteva serveru. Može da koristi običan VTable (to jest, element iz deklaracije tipa interfejsa), ili može da koristi dispinterface (interfejs za prosleđivanje). U osnovi, dispinterface je način za mapiranje svakog elementa interfejsa u broj. Pozivi serveru se zatim mogu prosleđivati po broju. Ovo možemo smatrati međukorakom između prosleđivanja naziva funkcije i upotrebe direktnog poziva VTable.

NAPOMENA

Termin dispinterface je zapravo ključna reč. Editor Type Library automatski generiše dispinterface za svaki interfejs. Pored dispinterface, Delphi koristi i druge ključne reči: dispid označava broj koji je dodeljen svakom elementu; readonly i writeonly su opcioni specifikatori svojstava. ■

Termin koji se koristi da bi se opisala ova mogućnost povezivanja sa serverom na dva različita načina, upotrebom više dinamičkog ili više statičkog pristupa, je dualni interfejs (dual interface). To znači da prilikom pisanja OLE kontrolera možete da odaberete da metodima servera pristupite na dva načina: možete da koristite kasno povezivanje i mehanizam koji obezbeđuje dispinterface, ili možete da upotrebite rano povezivanje i mehanizam koji se zasniva na VTable, tipovima interfejsa.

Važno je imati na umu (pored ostalih stvari) da različite tehnike daju brže ili sporije izvršavanje. Pronalaženje funkcije po nazivu (i izvršavanje provere tipova u vreme izvršavanja) je najsporiji

pristup, upotreba dispinterface je mnogo brži pristup, a upotreba direktnih poziva VTable je najbrži pristup. Ovakvu vrstu testa ćemo obaviti u primeru TlibCli, kasnije u ovom poglavlju.

Pisanje OLE Automation servera

Počećemo pisanjem OLE Automation servera. Da biste kreirali OLE Automation objekat, možete upotrebiti Delphijev Automation Object Wizard. Jednostavno počnite novu aplikaciju, otvorite Object Repository tako što ćete odabrati File➥New, preći na stranu ActiveX i odabrati Automation Object. U Automation Object Wizardu (prikazanom na slici 16.2) unesite naziv klase (bez početnog slova T, jer se ono dodaje automatski) i kliknite OK. Delphi će sada otvoriti editor Type Library.

SLIKA 16.2 *Delphijev Automation Object Wizard*

Kao što možete videti na slici 16.2, Delphi može da generiše OLE Automation servere koji takođe izvoze i događaje. Jednostavno potvrdite odgovarajuće polje Automation Object Wizarda, a Delphi će dodati odgovarajuće elemente u Type Library i u izvorni kod koji se generiše.

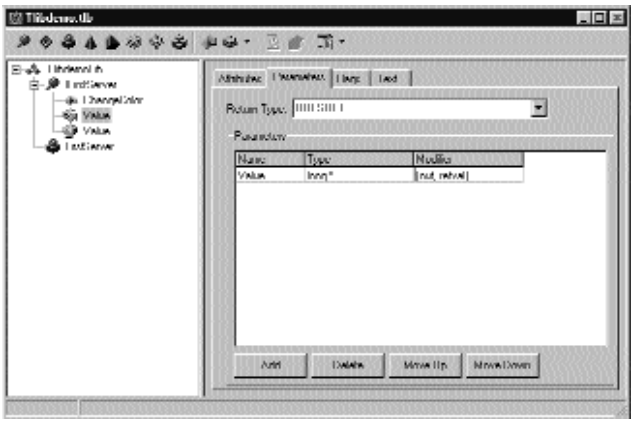
Editor Type Library

Editor Type Library je alat koji možete da koristite za definisanje Type Library u Delphiju. Na slici 16.3 je prikazan prozor editora Type Library pošto sam ja dodao neke elemente. Editor Type Library Vam omogućava da dodate metode i svojstva OLE Automation serveru koji smo upravo kreirali. Kada se to obavi, editor Type Library može da generiše Type Library (TLB) fajl i odgovarajući izvorni kod Object Pascala.

Da bismo izradili jednostavan primer, serveru možemo da dodamo svojstvo i metod. U editoru, zapravo, dodajemo ova dva elementa interfejsu, koji bi trebalo da bude nazvan `IFirstServer`. Jednostavno ga selektujte, a zatim kliknite kontrolu Method na paleti alata. (Nazivi ovih kontrola se mogu prikazati upotrebom lokalnog menija palete alata.) Sada je potrebno da metodu dodelite naziv, npr. `ChangeColor`. Naziv možete uneti u kontroli Tree View koja se nalazi na levoj strani prozora ili u polju Name koje se nalazi na desnoj strani. Delphi automatski definiše novi metod kao funkciju u polju Invoke Kind i (kao što ćete videti na strani Parameters) dodeljuje mu vrednost `HRESULT` koju daje, a ne dodeljuje nikakve parametre. Ovo odgovara Pascal definiciji:

```
procedure ChangeColor; safecall;
```

Postoje dva razloga za ovu razliku u tipu metoda. Prvi je da su u IDL jeziku, koji koristi COM, svi metodi naznačeni kao funkcije (čime se poštuje stil jezika C); drugi je da Delphi automatski obrađuje HRESULT kodove grešaka u svakom metodu koji koristi konvenciju pozivanja safecall.



SLIKA 16.3 *Editor Type Library koji prikazuje detalje interfejsa*

NAPOMENA

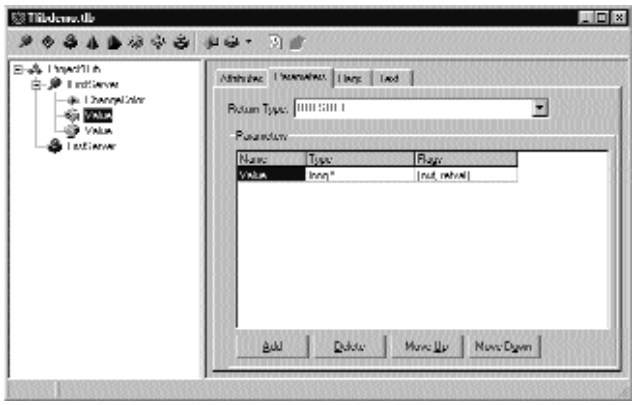
Metodi koji se nalaze u OLE Automation interfejsima u Delphiju generalno koriste konvenciju pozivanja `safecall`. Na ovaj način se postavlja `try-except` blok oko svakog metoda i obezbeđuje se unapred određena povratna vrednost koja označava uspeh ili grešku. ■

Sada možemo da dodamo svojstvo interfejsu tako što ćemo kliknuti kontrolu Property na paleti alata editora Type Library. Ponovimo, možete da unesete naziv svojstva u polje, npr. `Value`, i možete da odaberete tip podataka u combo polju Type. Pored izbora jednog od mnogih tipova podataka koji su već obezbeđeni, možete direktno uneti i druge tipove, naročito interfejse drugih objekata. Imajte na umu da OLE Automation podržava samo podskup Delphi tipova. U ovom primeru ćemo odabrati tip podataka `Long` koji odgovara Delphijevom tipu `Integer`.

Ukoliko ponovo pogledate stranu Parameters ovog primera (videti sliku 16.4), možete videti da metodi `Set` i `Get` (zapravo su nazvani `Put` i `Get` u COM žargonu) imaju povratnu vrednost `HRESULT`.

Takođe, dok metod `Put` koristi tip podataka svojstva kao svoj parametar (kao što je slučaj i sa Delphi svojstvima), metod `Get` koristi pokazivač na tip kao svoj parametar out. Ova definicija odgovara sledećim elementima Pascal interfejsa:

```
function Get_Value: Integer; safecall;
procedure Set_Value (Value: Integer); safecall;
property Value: Integer read Get_Value write Set_Value;
```



SLIKA 16.4 *Strana Parameters editora Type Library*

Kada kliknete kontrolu Refresh na paleti alata editora Type Library, generiše se Pascal verzija interfejsa. Ukratko ćemo je razmotriti, ali najpre želim da Vam usmerim pažnju na stranu Text ovog editora, koja sadrži definiciju koju smo upravo kreirali, a koja je napisana IDL jezikom:

```
interface IFirstServer: IDispatch
{
  [id(0x0000001)]
  HRESULT _stdcall ChangeColor ( void );
  [propget, id (0x0000002)]
  HRESULT _stdcall Value ( [out, retval] long * Value );
  [propput, id(0x0000002)]
  HRESULT _stdcall Value ( [in] long Value );
};
```

Na sreću, Delphijev editor Type Library Vas oslobađa pisanja sličnog koda, a opcije Delphi okruženja (na strani Type Library) sadrže opcionu kontrolu kojom birate Pascal ili IDL u tekstu koji prikazuje editor Type Library.

Kod servera

Sada možemo da zatvorimo editor Type Library i sačuvamo izmene. Ovom operacijom se dodaju tri elementa projektu: Type Library fajl, odgovarajuća Pascal definicija i deklaracija server objekta. Type Library je povezan sa projektom upotrebom iskaza uključivanja resursa, koji je dodat izvornom kodu fajla projekta:

```
{$R *.TLB}
```

U svakom trenutku možete ponovo otvoriti editor Type Library upotrebom komande View➥Type Library ili izborom odgovarajućeg TLB fajla iz normalnog Delphijevog okvira za dijalog File Open.

Kao što je ranije istaknuto, Type Library je takođe konvertovan u definiciju interfejsa i dodat je novoj Pascal jedinici. Ova jedinica je prilično duga, tako da sam ja u knjizi prikazao samo ključne elemente. Najvažniji deo je deklaracija novog interfejsa:

```
type
  IFirstServer = interface (IDispatch)
    ['{89855B42-8EFE-11D0-98D0-444553540000}']
    procedure ChangeColor; safecall;
    function Get_Value: Integer; safecall;
    procedure Set_Value (Value: Integer); safecall;
    property Value: Integer read Get_Value write Set_Value;
  end;
```

Zatim dolazi dispinterface, koji pridružuje broj svakom elementu interfejsa `IFirstServer`:

```
type
  IFirstServerDisp = dispinterface
    ['{89855B42-8EFE-11D0-98D0-444553540000}']
    procedure ChangeColor; dispid1;
    property Value: Integer dispid 2;
  end;
```

Poslednji deo fajla sadrži takozvanu klasu CoClass (koja se takođe prikazuje u editoru Type Library), koja se koristi za kreiranje objekta na serveru (te se iz ovog razloga koristi na klijent strani aplikacije, ne na server strani):

```
type
  CoFirstServer = class
  class function Create: IFirstServer;
  class function CreateRemote (
    const MachineName: string): IFirstServer;
  end;
```

Sve deklaracije ovog fajla (postoje i druge deklaracije koje sam izostavio) mogu se smatrati internom, sakrivenom podrškom implementaciji. Nije potrebno da ih u potpunosti razumete da biste napisali OLE Automation aplikacije.

Na kraju, Delphi generiše fajl sa deklaracijom aktuelnog objekta. Ova jedinica se dodaje aplikaciji i to je jedinica sa kojom ćemo raditi da bismo dovršili program. Ova jedinica deklariše klasu server objekta, koja mora da implementira interfejs koji smo upravo definisali:

```
type
  TFirstserver = class (TAutoObject, IFirstServer)
  protected
    function Get_Value: Integer; safecall;
    procedure ChangeColor; safecall;
    procedure Set_Value (Value: Integer); safecall;
  end;
```

Delphi nam već obezbeđuje osnovnu strukturu koda metoda, te je samo potrebno da popunimo linije koje se nalaze između. Sledi konačni kod metoda server objekta primera TLibDemo:

```
function TFirstServer.Get_Value: Integer;
begin
  Result := ServerForm.Value;
end;

procedure TFirstServer.ChangeColor;
begin
```

```
  ServerForm.ChangeColor;
end;

procedure TFirstServer.Set_Value (Value: Integer);
begin
  ServerForm := Value;
end;
```

U ovom slučaju, tri metoda se referišu na svojstvo i dva metoda koja sam dodao formularu. U opštem slučaju, ne smete da dodate kod koji se odnosi na korisnički interfejs unutar klase server objekta. Bolje je da se referišete na element korisničkog interfejsa, kao što je klasa formulara, i da zatim izvršite akcije.

Ja sam formularu dodao svojstvo jer sam želeo da promenim svojstvo Value i da dobijem sporedni efekat (da prikažem vrednost u polju za izmene). Server objekat, u ovom primeru, jednostavno prikazuje neka svojstva i metode aplikacije. Sledi deo deklaracije klase TServerForm koju sam ručno izmenio:

```
type
  TServerForm = class (TForm)
  ...
  private
    CurrentValue: Integer;
  protected
    procedure SetValue (NewValue: Integer);
  public
    property Value: Integer
      read CurrentValue write SetValue;
    procedure ChangeColor;
  end;
```

Implementacija ovih metoda je prilično direktna, što lako možete pogoditi kada pogledate njihov kod. Ono što je važno je metod SetValue koji može proizvesti sporedni efekat:

```
procedure TServerForm.SetValue (NewValue: Integer);
begin
  if NewValue <> CurrentValue then
  begin
    CurrentValue := NewValue;
    UpDown1.Position := Current.Value;
  end;
end;
```

Formular ovog primera sadrži polje za izmene sa pridruženom komponentom UpDown kao i nekoliko kontrola za prikazivanje aktuelne vrednosti i za promenu boje. Formular u vreme dizajniranja možete videti na slici 16.5.

SLIKA 16.5 *Formular primera TLibDemo u vreme dizajniranja*

Registrovanje Automation servera

Jedinica koja sadrži server objekat ima još jedan iskaz koji Delphi dodaje odeljku initialization:

```
initialization
  TAutoObjectFactory.Create (ComServer, TFirstServer,
    Class_FirstServer, ciMultiInstance);
end.
```

NAPOMENA

U ovom slučaju, ja sam odabrao višestruko instanciranje. Za različite stilove instanciranja u COM-u pogledajte dodatak "COM modeli instanciranja i više linija" u Poglavlju 15. ■

Ovo se ne razlikuje mnogo od kreiranja radionica klasa koje smo videli u primerima prethodnog poglavlja. Kombinovan sa pozivom metoda Intialize objekta Application, koji Delphi po definiciji dodaje izvornom kodu bilo kojeg programa projekta, prethodni kod initialization čini registraciju servera direktnom.

Informacije servera možete dodati Windows Registryju izvršavanjem ove aplikacije na odredišnoj mašini (kompjuter na koji želite da instalirate OLE Automation server), prosleđujući mu parametar `/regserver` sa komandne linije. Isto možete učiniti ukoliko odaberete Start➥Run, upotrebom Explorera ili File Managera, ili izvršavanjem programa u okviru Delphija pošto ste uneli parametar komandne linije (upotrebom komande Run➥Parameters). Drugi parametar komandne linije, parametar `/unregserver`, koristi se za uklanjanje ovog servera iz Registryja.

Pisanje klijenta za naš server

Sada kada smo izradili server, možemo da pripremimo jednostavan klijent program da bismo testirali server. Ovaj klijent se može povezati sa serverom upotrebom promenljivih ili upotrebom novog Type Libraryja. Drugi pristup se može implementirati ručno ili upotrebom novih Delphi 5 tehnika za obavijanje komponenata oko Automation servera. Mi ćemo isprobati sve ove pristupe.

Kreirajte novu aplikaciju — ja sam je nazvao TlibCli — a zatim otvorite Type Library fajl servera, pošto ga (opciono) kopirate u direktorijum projekta. Sačuvajte Type Library fajl, upotrebivši Delphijevu komandu menija File➥Save, i nova verzija deklaracija interfejsa će biti generisana za

Vas. Naravno, u ovom slučaju mogli ste da preuzmete Pascal deklaracije iz izvornog koda servera, ali ja se trudim da poštujem opšti pristup, koji se takođe može primeniti kada još uvek niste napisali server. Zapravo, obično možete da izdvojite Type Library direktno iz izvršnog fajla servera ili iz DLL-a koji se dobija uz program.

UPOZORENJE

Nemojte da dodajete Type Library klijent aplikaciji jer pišemo OLE Automation kontroler, a ne server. Delphi projekat kontrolera ne uključuje Type Library servera na koji se povezuje. ■

Možete se jednostavno referisati na Pascal fajl koji je generisao editor Type Library u kodu glavnog formulara:

```
uses
  TlibdemoLib_TLB;
```

Već sam istakao da je jedan od elemenata ove jedinice koji je generisao Type Library klasa *kreiranja* (creation class), ili CoClass, specijalna klasa sa dve klasne funkcije koje možete koristiti za lokalno kreiranje objekta servera ili za udaljeno kreiranje (upotrebom DCOM-a). Već sam Vam prikazao interfejs ove klase, a ovde prikazujem njenu implementaciju:

```
class function CoFirstServer.Create: IFirstServer;
begin
  Result := CreateComObject (Class_FirstServer)
    as IFirstServer;
end;

class function CoFirstServer.CreateRemote (
  const MachineName: string): IFirstServer;
begin
  Result := CreateRemoteComObject (MachineName,
Class_FirstServer) as IFirstServer;
end;
```

Prvu od ove dve funkcije, funkciju `Create`, možete da upotrebite za kreiranje server objekta (i možda da pokrenete server aplikaciju) na istom kompjuteru. Drugu funkciju, funkciju `CreateRemote`, možete upotrebiti za kreiranje servera na nekom drugom kompjuteru, ukoliko Vaša verzija operativnog sistema podržava DCOM.

Dve funkcije jednostavno predstavljaju prečicu za poziv `CreateComObject` koji Vam omogućava da kreirate instancu COM objekta ukoliko znate njegov GUID. Kao alternativu možete, takođe, upotrebiti funkciju `CreateOleObject` koja kao parametar zahteva registrovani naziv servera. Postoji još jedna razlika između ove dve funkcije za kreiranje: `CreateComObject` kao rezultat daje objekat tipa `IUnknown`, dok `CreateOleObject` kao rezultat daje objekat tipa `IDispatch`.

U svom primeru ću koristiti `CoFirstServer.Create`. Kada kreirate server objekat, kao povratnu vrednost možete dobiti interfejs `IFirstServer`. Možete ga koristiti direktno ili ga možete sačuvati u promenljivoj tipa variant. Evo primera prvog pristupa:

```
var
  MyServer: Variant;
begin
   MyServer := CoFirstServer.Create;
  MyServer.ChangeColor;
```

Ovaj kod, koji se zasniva na tipu variant, ne razlikuje se mnogo od koda prvog kontrolera koji smo izradili u ovom poglavlju (kontroler koji je koristio Microsoft Word). Evo i alternativnog koda koji proizvodi identičan efekat:

```
var
  IMyServer: IFirstServer;
begin
   IMyServer := CoFirstServer.Create;
  IMyServer.ChangeColor;
```

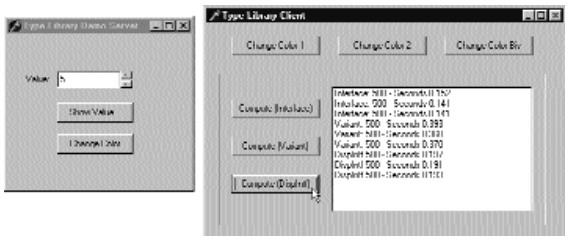
Interfejsi, promenljive i Dispatch interfejsi: testiranje razlika u brzini

Kao što sam istakao u uvodnom odeljku o bibliotekama tipa, jedna od razlika između ovih pristupa je brzina. Prilično je komplikovano prikazati tačne performanse svake od tehnika, jer na brzinu utiče mnogo faktora. Ja sam primeru TLibCli dodao jednostavan test samo da bih Vam dao ideju. Sledi kod testa, petlja koja pristupa svojstvu `Value` servera. Kompletna vrednost se prikazuje samo da bi se zavarao optimizator koji bi inače mogao da ukloni deo koda. Pravi izlaz programa se odnosi na merenje vremena koje se određuje pozivanjem API funkcije `GetTickCount` pre i posle izvršavanja petlje. (Postoje dve mogućnosti: da koristite Delphijeve funkcije za merenje vremena, koje su nešto manje precizne, ili da koristite veoma precizne funkcije za merenje vremena jedinice za multimedijalnu podršku, jedinice MMSystem.) Evo koda jednog od metoda; metodi su veoma slični:

```
procedure TClientForm.BtnIntfClick (Sender: TObject);
var
  I, K: Integer;
  Ticks: Cardinal;
begin
  Screen.Cursor := crHourglass;
  try
    Tiskc := GetClickCount;
    K := 0;
    for I := 1 to 100 do
      K := K + IMyServer.Value;
    Ticks := GetTickCount — Tick;
    ListResult.Items.Add (Format (
      'Interface: %d — Seconds %.3f', [K, Ticks / 1000]));
  finally
    Screen.Cursor := crDefault;
  end;
end;
```

Ovim programom možete da poredite izlaz dobijen pozivanjem ovog metoda koji se zasniva na interfejsu, odgovarajuće verzije koja se zasniva na promenljivoj, pa čak i treće verzije koja se zasniva na interfejsu prosleđivanja. Primer izlaza (koji je dodat listi da biste mogli da obavite nekoliko testiranja i da uporedite rezultate) prikazan je na slici 16.6. Očigledno je da vreme izvršavanja zavisi od

brzine Vašeg kompjutera, a i rezultate možete promeniti povećavanjem ili smanjivanjem maksimalne vrednosti brojača petlje.



SLIKA 16.6 *TLibCli OLE Automation kontroler može pristupiti serveru na različite načine koji daju različite rezultate performansi. Primetićete da se prozor servera nalazi u pozadini*

Već smo videli kako možete koristiti interfejs i promenljivu. Šta se dešava sa interfejsom prosleđivanja? U ovom slučaju možete jednostavno deklarisati promenljivu tipa interfejsa prosleđivanja:

```
var
  DMyServer: IFirstServerDisp;
```

Zatim je možete upotrebiti da biste pozvali metode na uobičajen način, pošto ste joj dodelili objekat konvertovanjem objekta koji ste dobili od klase CoClass:

```
DMyServer := CoFirstServer.Create as IFirstServerDisp;
```

Kada pogledate izmereno vreme i interni kod primera, očigledno je da postoje veoma male razlike između upotrebe interfejsa i prosleđivanja interfejsa jer su oni zapravo međusobno povezani. Drugim rečima, možemo reći da su interfejsi prosleđivanja tehnika između interfejsa i promenljivih, ali nam gotovo u potpunosti daju brzinu koja je jednaka interfejsima.

Oblast delovanja Automation objekata

Drugi važan element koji treba imati na umu jeste opseg delovanja Automation objekata. Promenljivi i interfejs objekti koriste tehnike prebrojavanja referenci, te ukoliko je promenljiva koja se odnosi na interfejs objekat deklarisana lokalno u metodu, na kraju metoda objekat će biti uklonjen i može se desiti da se prekine izvršavanje servera (ukoliko su uklonjeni svi objekti koje je kreirao server). Na primer, pisanje metoda kodom kakav je sledeći kod, proizvodi veoma malo efekta:

```
procedure TClientForm.ChangeColor;
var
tMyServer: IFirstServer;
begin
IMyServer := CoFirstServer.Create;
IMyServer ChangeColor;
end;
```

Izuzev ukoliko server već nije aktivan, kreira se kopija programa i menja se boja, ali se odmah potom zatvara server, jer objekat interfejs tipa izlazi iz opsega. Alternativni pristup koji sam koristio u primeru TLibCli je deklarisanje objekta kao polja formulara i kreiranje COM objekata prilikom pokretanja, kao u sledećoj proceduri:

```
procedure TClientForm.FormCreate (Sender: TObject);
begin
IMyServer := CoFirstServer.Create;
end;
```

Ovim kodom, kada se pokrene klijent program, odmah se aktivira server program. Kada program završi izvršavanje, polje formulara se uklanja i server se zatvara. Druga mogućnost je deklarisanje objekta u formularu, ali ga tada kreirajte samo kada se koristi kao u sledećim fragmentima koda:

```
// MyServerBis: Variant;
if varPype (MyserverBis) = varEmpty then
MyServerBis := CoFirstServer.Create;
MyServertBis.ChangeColar;

// IMyServerBis: IFirsrServer;
if not Assigned (IMyServerBis) then
IMyServerBis := CoFirstServer.Create;
IMyServerBis.ChangeColor;
```

NAPOMENA

Promenljiva se inicijalizuje u tip `varEmpty` kada se kreira. Ukoliko umesto toga dodelite vrednost null, promenljive postaju tipa `varNull`. I `varEmpty` i `varNull` predstavljaju promenljive kojima nije dodeljena vrednost, ali se drugačije ponašaju u izrazima. Vrednost `varNull` prolazi kroz izraz (čineći izraz null izrazom), dok `varEmpty` potpuno nestaje. ■

Server u komponenti

Prilikom kreiranja klijent programa, za naš server ili bilo koji drugi Automation server možemo upotrebiti novi Delphi 5 pristup, drugim rečima, možemo oko COM servera obmotati komponentu. Zapravo, ukoliko pogledate poslednji odeljak `TlibdemoLib_TLB` fajla, pronaći ćete sledeće deklaracije:

```
// OLE Server Proxy class declaration
TFirstServer = class(ToleServer)
private
FIntf: IFirstServer;
FProps: TFirstServerPropertien;
function GetServerPropertias: IFirstServerProperties;
function GetDefaultIterface: IFirstServer;
protected
procedure InitServerData; override;
function Get_Value: Integer;
procedure Set_Value(Value: Integer);
public
constructor Create(AOwner: TComponerit); override;
destructor Destroy: override;
procedure Connect: override:
procedure (ConnectTo(svrIntf: IFirstServer);
procedure Disconnect; override;
procedure ChangeColor;
property DefaultInterface: IFirstServer
read GetDefaultInterface;
property Value: Integer
```

```
    read Get_Value write Set_Value;
  published
  property Server: TFirsrServerProperties
  read GetServerProperties;
  end;
```

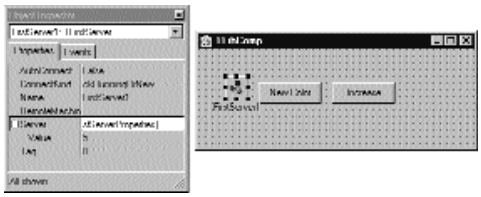
Ovo je nova komponenta izvedena iz klase TOleServer koju sistem registruje u proceduri Register, koja je deo jedinice. Ukoliko ovu jedinicu dodate paketu, nova server komponenta će postati dostupna u Delphijevoj paleti Component Palette. Takođe, možete uvesti Type Library novog servera (upotrebom komande menija Project➥Import Type Library), dodati server listi (ukoliko kliknete kontrolu Add i odaberete izvršni fajl servera) i instalirate je u novom ili postojećem paketu. Komponenta će biti smeštena na stranu Servers Component Palette. Okvir za dijalog Import Type Library, koji naznačava ove operacije, možete videti na slici 16.7.

SLIKA 16.7 *Okvir za dijalog Import Type Library se može koristiti za uvoženje Automation server objekta kao nove Delphi komponente*

Ja sam kreirao novi paket, paket AutoPack, koji možete pronaći u direktorijumu projekta TlibDemo. U ovaj paket sam dodao direktivu **LIVE_SERVER_AT_DESIGN_TIME** na strani Directories/Conditionals okvira za dijalog Project Options paketa. Ovim se aktivira dodatna karakteristika koju ne dobijate po definiciji: u vreme dizajniranja server komponenta će imati dodatno svojstvo koje kao podelemente izlistava sva svojstva Automation servera. Primer koji je preuzet u vreme dizajniranja iz primera `TLibComp`, možete videti na slici 16.8.

UPOZORENJE

Direktivu **LIVE_SERVER_AT_DESIGN_TIME** treba pažljivo upotrebljavati kada se koristi uz najkomplikovanije Automation servere (uključujući programe kakvi su Word, Excel, PowerPoint i Visio). Zapravo, ovo podešavanje zahteva da se aplikacije nalaze u određenom modu pre nego što možete da upotrebite neka svojstva njihovih Automation interfejsa. Na primer, desiće se izuzeci ukoliko pristupite Word serveru pre nego što se u Wordu otvori dokument. To je razlog zbog kojeg ova karakteristika u Delphiju nije po definiciji aktivna — problematična je u vreme dizajniranja za mnoge servere. ■



SLIKA 16.8 *Server komponenta, sa aktivnim svojstvima u vreme dizajniranja*

Kao što možete videti, Object Inspector pokazuje da komponenta sadrži nekoliko svojstava. Svojstvo `AutoConnection` označava kada treba pokrenuti server komponentu u vreme dizajniranja i čim započne izvršavanje klijent programa. Alternativa je da se pokrene Automation server prvi put kada se pozove neki od njegovih metoda. Drugo svojstvo, svojstvo `ConectKind`, označava kako da se uspostavi veza sa serverom. Ovo svojstvo uvek može da započne novu instancu (`ckNewInstance`), upotrebi instancu koja se izvršava (`ckRunningInstance`, koja dovodi do greške u pristupanju ukoliko se server već ne izvršava), odabere aktulenu instancu ili pokrene novu ukoliko nijedna nije dostupna (`ckRunningOrNew`). Konačno, možete da zatražite udaljeni server upotrebom `ckRemote` i direktno se povežete sa serverom u kodu posle ručnog povezivanja upotrebom `ckAttachToInterface`.

OLE tipovi podataka

OLE i COM ne podržavaju sve tipove podataka koje imate na raspolaganju u Delphiju. Ovo je naročito važno za OLE Automation jer se klijent i server često izvršavaju u zasebnim adresnim prostorima, a sistem mora da prebacuje podatke s jedne strane na drugu. Takođe, imajte na umu da bi OLE interfejsima trebalo da bude moguće pristupati iz programa napisanih u bilo kom programskom jeziku.

COM tipovi podataka uključuju osnovne tipove podataka kao što su Integer, SmallInt, Byte, Single, Double, WideString, Variant i WordBool (ali ne i Boolean). Sledi mapiranje nekih osnovnih tipova podataka, koji su na raspolaganju u editoru Type Library, prema odgovarajućim Delphi tipovima podataka:

OLE tip podataka	Delphi tip podataka
`BSTR`	`WideString`
`byte`	`ShortInt`
`CURRENCY`	`Currency`
`DATE`	`TDateTime`
`DECIMAL`	`TDecimal`
`double`	`Double`
`float`	`Single`
`GUID`	`GUID`
`int`	`SYSINT`
`long`	`Integer`
`LPSTR`	`PChar`

OLE tip podataka	Delphi tip podataka
LPWSTR	PWideChar
short	SmallInt
unsigned char	Byte
unsigned int	SYSUINT
unsigned long	UINT
unsigned short	Word
VARIANT	OleVariant

Primetićete da je SYSINT trenutno definisan kao Integer, te se nemojte sekirati zbog izgleda čudne definicije tipa. Pored osnovnih tipova podataka možete, takođe, koristiti OLE tipove podataka za složene elemente, kao što su fontovi, liste stringova i bitmape, koristeći interfejse IFontDisp, IStrings i IPictureDisp. Naredni odeljci opisuju detalje servera koji obezbeđuje listu stringova i font za klijenta.

Isticanje lista stringova i fontova

Primer ListServ je praktična demonstracija toga kako možete da istaknete dva složena tipa kao što su stringovi i font, iz OLE Automation servera koji je napisan u Delphiju. Ja sam odabrao ova dva specifična tipa jer su oba podržana u Delphiju.

Windows obezbeđuje interfejs IFontDisp koji je dostupan i u ActiveX jedinici. Delphi jedinica AxCtrls proširuje ovu podršku obezbeđivanjem metoda konverzije, kao što su metodi GetOleFont i SetOleFont. Delphi obezbeđuje interfejs IStrings u StdVCL jedinici, a AxCtrls jedinica obezbeđuje funkcije konverzije za ovaj tip (kao i za treći tip koji ću koristiti, tip TPicture).

Server koji izrađujemo sadrži veoma jednostavan formular na kojem se nalazi komponenta ListBox. Formular sadrži Automation objekat izrađen oko sledećeg interfejsa:

```
type
  IListServur = interface (Idispatch)
    ['(323C4A84-E400-11DI-B9F1-004845400FAA)']
    function Get_Items: IString: safecall;
    procedure Set_Items (const Value: IStrings); safecall;
    function Get_Font: IFontDisp; safecall;
    procedure Set_Font(const Value: IFontDisp): safecall:
    property Items: IStrings read Get_Items write Set_Items;
    property Font: IFontDisp read Get_Font write Set_Font;
  end;
```

Server objekat sadrži ista četiri metoda koja su izlistana u njegovom interfejsu, kao i neke privatne podatke koji služe za čuvanje statusa, funkcije inicijalizacije i destruktora:

```
type
  TListServer = class (TAutoObject, IListServer)
  private
    fItems: TStrings;
    fFont: TFont;
  protected
    function Get_Font: IFontDisp: safecall;
```

```
    function Get_Items: IStrings: safecall;
    procedure Sem_Font (const Value: IFontDisp); safecall;
    procedure Set_Items (Const Value: IStrings); safecall;
  public
    destructor Destroy; override;
    procedure Initialize; override;
  end;
```

Kod metoda je, zapravo, veoma jednostavan. Pseudokonstruktor kreira interne objekte, a destruktor ih uklanja. Evo prvog:

```
procedure TFistServer.Initialize;
begin
  inherited Thitialize;
  fItems :=  TStringList.Create;
  fFont := TFontCreate;
end;
```

Metodi Set i Get su, takođe, veoma jednostavni. Ovi metodi kopiraju informacije iz OLE interfejsa u lokalne podatke, a odatle na formular i obrnuto. Sledi kod dva metoda stringova (druga dva, za font, su slični te ih ovde neću prikazati):

```
function TListServer.Get_Items: IStrings;
begin
  // get the listbox items, converting them
  GetOldStrings (ListServForm.Listbox1.Items, Result);
end;

procedure TListServer.Set_Items(const Value: IStrings);
begin
  // convert the strings, received as parameter
  SetOleStrings (ListServForm.ListBox1.Items, Value);
end;
```

Kada smo kompajlirali i registrovali server, možemo da obratimo pažnju na klijent aplikaciju. Ovo jednostavno ugnežđuje Pascal prevođenje Type Library servera, kao u prethodnom primeru, a zatim implementira objekat koji koristi interfejs.

Umesto kreiranja servera prilikom pokretanja objekta, klijent program kreira server kada je server potreban. Ja sam ovu tehniku ranije opisao, ali problem je u tome što postoji nekoliko kontrola koje korisnik može da klikne, a pošto ne želimo da namećemo bilo kakakv redosled, svaki događaj treba da ima obradu kakva je sledeća:

```
if not Assigned (ListServ) then
  ListServ := CoListServer.Create;
```

Ovakvo dupliranje koda je prilično opasno, te sam ja odlučio da upotrebim drugačiji pristup. Definisao sam svojstvo koje odgovara interfejsu servera i za interfejs servera sam definisao metod za čitanje. Svojstvo je mapirano na neke interne podatke koje sam definisao pod različitim nazivom da bih izbegao grešku direktne upotrebe. Evo definicija koje sam dodao klasi formulara:

```
private
  fInternalListServ: IListServer;
  function GetListSrv: IListServer;
public
```

```
  property ListSrv: IListServer;
    read GetListSrv;
```

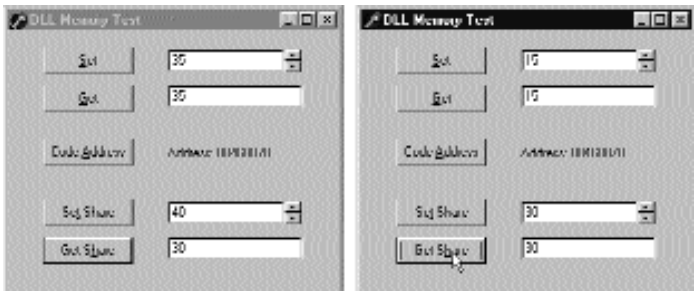
Implementacija metoda Get može da proveri da li objekat već postoji. Ovaj kod će se često ponavljati, ali to ne bi trebalo da primetno uspori aplikaciju:

```
function TListCliForm.btnFontClick (Sender: TObject);
begin
  // eventually create the server
  if not Assigned (fInternalListServ) then
    fInternalListServ := CoListServer.Create;
  Result := fI InternalListServ;
end;
```

Ostatak koda klijent aplikacije je prilično jednostavan, a primer izvršavanja programa možete videti (pored servera) na slici 16.9. Sledi primer selektovanja fonta, koji se zatim šalje serveru:

```
procedure TLictCliForm.btnFontClick(Sender: TObject);
var
  NewFont: IFontDisp:
begin
  // select a font and apply it
  if FontDialog1.Execute then
  begin
    GetOleFont (FontDialog1.Font, NewFont):
    ListSrv.Font := NewFont;
  end:
end;
```

Takođe, postoji nekoliko metoda koji se odnose na stringove, a koje možete videti kada pogledate izvorni kod programa.



SLIKA 16.9 *Aplikacije ListCli i ListSrv dele složene podatke, preciznije fontove i liste stringova*

Upotreba Office programa

Do sada smo izradili i klijent i server stranu OLE Automation konekcije. Ukoliko je Vaš cilj da samo omogućite da dve aplikacije (koje ste izradili) komuniciraju, ovo je svakako korisna tehnika, mada nije i jedina. Do sada smo videli neke različite pristupe deljenja podataka u prethodna dva poglavlja (upotrebom memorijski mapiranih fajlova i upotrebom poruke `wm_CopyData`). Prava vrednost OLE Automationa je da je to standard, te ga možete upotrebiti da integrišete Vaše Delphi programe sa ostalim aplikacijama koje posedujete. Tipičan primer integracije programa je integracija sa Office programima kao što su Microsoft Word i Microsoft Excel, ili čak i sa samostalnim aplikacijama kakva je AutoCAD.

Integracija sa ovim aplikacijama donosi dvostruku prednost:

- Možete dopustiti korisnicima da rade u okruženju koje poznaju, na primer, da generišu izveštaje i memorandume na osnovu podataka iz baze podataka u formatu kojim lako mogu da manipulišu.
- Integracija Vam omogućava da izbegnete implementiranje složene funkcionalnosti počevši od nule, kao što je pisanje Vašeg koda za obradu teksta unutar programa. Umesto da samo ponovo koristite jednostavne komponente, možete da koristite složene aplikacije.

Postoje i neki nedostaci u ovom pristupu, koje svakako vredi pomenuti:

- Korisnik mora da poseduje aplikacije sa kojima želite da izvršite integraciju, a možda je potrebna poslednja verzija aplikacije da bi bile podržane sve karakteristike koje koristite u Vašem programu.
- Potrebno je da naučite novi programski jezik i programsku strukturu, a često ćete biti ograničeni literaturom koju posedujete. Istina je, naravno, da još uvek koristite Pascal, ali kod koji pišete zavisi od OLE tipova podataka, tipova koje uvodi server, a naročito od klasa koje je obično teško razumeti.
- Na kraju možete dobiti aplikaciju koja funkcioniše samo sa specifičnom verzijom server aplikacije, naročito ukoliko pokušate da optimizujete pozive upotrebom interfejsa umesto promenljivih. Preciznije, Microsoft ne nastoji da održi kompatibilnost skriptova između novih verzija Worda i drugih Office aplikacija.

Već smo videli mali deo koda primera WordTest, ali sada želim da kompletiram ovaj jednostavan ali interesantan test program dodavanjem nekoliko novih karakteristika.

Slanje podataka Microsoft Wordu

Delphi 5 je pojednostavio upotrebu Microsoft Office aplikacija preinstaliranjem nekih komponenata koje obavijaju Automation interfejs ovih servera. Ove komponente, koje su dostupne na Servers strani Component Palette, instalirane su upotrebom istih tehnika koje sam pokazao u prethodnom pdeljku. Ono što želim da istaknem je da prava Delphi 5 inovacija leži u tehnici kreiranja komponenata koje obavijaju postojeće Automation servere, pre nego u mogućnostima unapred određenih server komponenata.

Tehnički je moguće koristiti promenljive uz Automation servere, kao što smo videli u odeljku "Uvod u biblioteke tipa (Type Libraries)". Upotreba interfejsa i biblioteka tipa je svakako bolja jer Vam kompajler pomaže da pronađete greške u izvornom kodu i proizvodi brži kod. Zahvaljujući novoj server komponenti ovaj proces je prilično jednostavan.

Ja sam napisao program, koji sam nazvao DBOffice, koji koristi unapred određene komponente Delphija 5 za slanje tabele Wordu i Excelu. U oba slučaja možete da koristite objekat aplikacije, objekat dokumenta/radne tabele, ili njihovu kombinaciju. Postoje i druge specijalizovane komponente, za zadatke kakav je rukovanje Excel grafikonima, ali ovaj primer će biti dovoljan za predstavljanje ugrađenih Office komponenata.

Kada je u pitanju Microsoft Word, ja koristim samo objekat dokumenta sa unapred određenim podešavanjima. Kod koji se koristi za slanje tabele Wordu počinje dodavanjem teksta dokumentu:

```
procedure TFormOff.BtnWordClick (Sender: TObject);
begin
  WordDocument1.Activate;
  // insert title
  WordDocument1.Range.Text := 'American Capitals from ' +
    Table1.TableName;
  WordDocument1.Range.Font.Size := 14;
```

Ovaj kod koristi tipičnu petlju while, koja pretražuje tabelu baze podataka, a sadrži sledeći kod:

```
while not Table1.EOF do
begin
  // send the two fields
  WordDocument1.Range.InsertParagraphAfter;
  WordDocument1.Paragraphs.Last.Range.Text :=
    Table1.FieldByName ('Name').AsString + #9 +
    Table1.FieldByName ('Capital').AsString;
  Table1.Next;
end;
```

Poslednji deo koda je nešto komplikovaniji. Ovaj deo koda radi sa selekcijom i redom tabele, koji se respektivno smeštaju u dve promenljive tipa `Range` i `Row`, koje definiše Word, a dostupne su iz Word 97 jedinice:

```
procedure TFormOff.BtnwordClick(Sender: TObject);
var
  RangeW: Word97. Range;
  v1: Variant;
  ov1: OleVariant;
  Row1: Word97.Row;
begin
  // code above...
  RangeW := WordDocumentt.Content;
  v1 := RangW;
  v1.ConvertToTable (#9, 19, 2);
  Row1 := WordDocument1.Tables.Item(1).Rows.Get_First;
  Row1.Range.Bold := 1;
  Row1.Range.Font.Size := 30;
  Row1.Range.InsertParagraphAfter;
  ov1 := ' ';
```

```
  Row1.ConvertToText (ov1);
end;
```

Kao što možete videti u poslednjem iskazu prethodnog koda, da biste prosledili parametar, najpre morate da sačuvate parametar u promenljivoj `OleVariant` jer se mnogi parametri prosleđuju po referenci, pa ne možete da prosledite konstantnu vrednost. Ovo nagoveštava da — ukoliko imate veliki broj parametara — morate da definišete broj parametara, iako Vam odgovaraju unapred određene vrednosti. Alternativa, koja se često koristi, jeste upotreba privremene promenljive tipa Variant na koju primenjujete metod, jer tip Variant ne zahteva striktnu proveru tipa parametara. Ova tehnika se koristi u prethodnom kodu za poziv metoda `ConvertToTable`, koji sadrži više od 10 parametara.

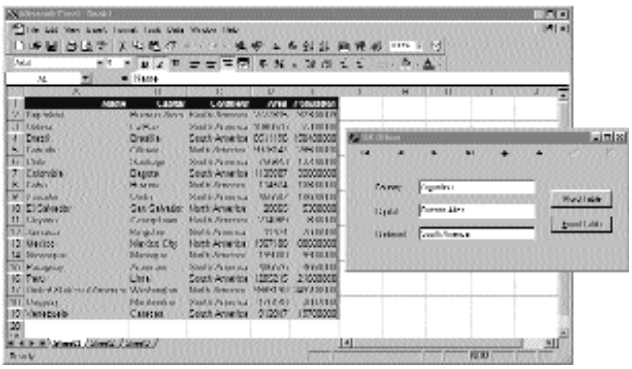
Izrada Excelove tabele

Kada je u pitanju Excel, koristio sam nešto drugačiji pristup i radio sam sa objektom aplikacije. Kod kreira novu radnu tabelu Excela, popunjava je tabelom baze podataka i formatira rezultat. Kod koristi Excelov interni objekat, objekat Range, koji ne treba pomešati sa sličnim tipom koji je na raspolaganju u Wordu (što je razlog za upotrebu prefiksa kod imena ovog tipa u jedinici koja definiše Excel Type Library). Evo kompletnog koda:

```
procedure TFormOff.BtnExcelClick(Sender: TObject);
var
  RangeE: Excel97.Range;
  I, Row: Integer;
  Bookmark: TBookmarkStr;
begin
  // create and show
  ExcelApplication1.Visible [0] := True;
  ExcelApplication1.Workbooks.Add (NULL, 0);
  // fill is the first row with field titles
  RangeE := ExcelApplication1.ActiveCell
  for I := 0 to Table1.Fields.Count — 1 do
  begin
    RangeE.Value := Table1.Fields [I] DisplayLabel;
    RangeE := RangeE.Next;
  end;
  // add field data in following rows
  Table1.DisableControls;
  try
    Bookmark := Table1.Bookmark;
    try
      Tablel. First;
      Row := 2;
      while not Table1.EOF do
      begin
        RangeE := ExcelApplication1.Range ['A' + IntToStr (Row),
          'A' + IntToStr (Row)];
        for I := D to Table1.Fields.Count - 1 do
        begin
          RangeE.Value := Table1.Fields [I].AsString;
          RangeE := RangeE.Next;
        end;
```

```
        Table1.Next;
        Inc (Row);
      end;
    finally
      Table1.Bookmark := Bookmark;
    end;
  finally
    Table1.EnableControls;
  end;
  // format the section
  RangeE := ExcelApplication1.Range [ 'A1', 'E' + IntToStr (Row - 1)];
  RangeE.AutoFormat (3, NULL, NULL, NULL, NULL, NULL, NULL);
end;
```

Efekat ovog koda možete videti na slici 16.10. Primetićete da ja u kodu ne obrađujem nijedan događaj Office aplikacija, ali mnogi događaji su Vam na raspolaganju. Obrađivanje ovih događaja je u prošlosti bilo veoma komplikovano, ali ih je sada jednostavno obraditi koliko i događaje Delphi komponenata. Prisustvo ovih komponenata je razlog da imamo specifične objekte za dokumente i druge specifične elemente: možda želite da znate kada je korisnik zatvorio dokument, te je zbog toga ovo događaj objekta dokumenta, a ne objekta aplikacije.



SLIKA 16.10 *Excelova radna tabela koju je generisala aplikacija DbOffice*

NAPOMENA

Kada koristite komponente Office servera, jedan od glavnih problema je nedostatak adekvatne dokumentacije. Mada Microsoft distribuira nešto dokumentacije uz krajnju verziju Office paketa, to svakako nije dovoljno za Delphi. Potpuno drugačiji pristup rešavanju problema je "Office Partner", skup komponenata koje obezbeđuje DeVries Data Systems, Inc. (`www.dvdata.com`). Ove komponente mapiraju Office servere, kao što to čine komponente koje imate na raspolaganju u Delphiju, ali pružaju obimne editore svojstava koji Vam omogućavaju da vizuelno radite sa internom strukturom ovih servera. Upotrebom ovih editora svojstava možete kreirati dokumente, paragrafe, tabele i sve druge interne objekte, čak i u vreme dizajniranja! Moje iskustvo je da ovo može da Vam uštedi dosta vremena. ■

Upotreba složenih dokumenata

Složeni dokumenti (Compound Documents), ili aktivni dokumenti (Active Documents), jesu Microsoftov naziv za tehnologiju koja na mestu (in-place) omogućava editovanje dokumenta iz nekog drugog dokumenta (na primer, sliku u Wordovom dokumentu). Ovo je tehnolgija koja je uvela termin OLE, ali — mada se još uvek koristi — njena uloga je definitivno ograničenija nego što je Microsoft predvideo kada ju je predstavio oko 1990. godine. Složeni dokumenti, zapravo, imaju dve različite mogućnosti — povezivanje objekata (object linking) i ugnežđavanje objekata (object embeding) — što daje termin OLE.

- Ugnežđavanje objekta u složeni dokument odgovara pametnoj verziji operacija kopiranja i unošenja koje činite kada koristite Clipboard. Ključna razlika je u tome da — kada kopirate OLE objekat iz server aplikacije i prenesete ga u kontejner aplikaciju, Vi kopirate podatke i neke informacije o serveru (njegov GUID). Ovo Vam omogućava da aktivirate server aplikaciju iz kontejnera da biste editovali podatke.
- Povezivanje objekta sa složenim dokumentom umesto toga kopira samo referencu na podatke i informacije o servereu. U opštem slučaju povezivanje objekta aktivirate upotrebom Clipboarda i vršenjem operacije Paste Link. Kada editujete podatke u kontejner aplikaciji, Vi zapravo menjate originalne podatke, koji se čuvaju u zasebnom fajlu.

Kako se server program referiše na ceo fajl (a samo deo može biti povezan sa klijent dokumentom), server će biti aktiviran u zasebnom prozoru i radiće sa celim originalnim fajlom, a ne samo sa podacima koje ste kopirali. Kada imate ugnežđen objekat, kontejner može podržati vizuelno editovanje (ili na mestu), što znači da objekat možete da menjate u kontekstu, unutar glavnog prozora kontejnera. Prozori server i kontejner aplikacija, njihovi meniji i njihove palete alata se automatski spajaju, omogućavajući korisniku da radi unutar jednog prozora sa velikim brojem različitih tipova objekata — te zbog toga i sa velikim brojem različitih OLE servera — a da ne napušta prozor kontejner aplikacije.

Druga ključna razlika između ugnežđavanja i povezivanja je ta da podatke ugnežđenog objekta čuva kontejner aplikacija i da kontejner aplikacija njima manipuliše. Kontejner aplikacija čuva ugnežđeni objekat unutar svog fajla. Nasuprot tome, povezani objekat se fizički nalazi u zasebnom fajlu, kojim manipuliše isključivo server, čak i kada se veza odnosi na mali deo fajla.

U oba slučaja, kontejnr aplikacija ne mora da zna kako da obradi objekat i njegove podatke — čak ni kako da ih prikaže — bez pomoći servera. Server aplikacija ima mnogo posla koji treba da obavi, čak i kada ne editujete podatke. Kontejner aplikacija često pravi kopiju slike OLE objekta i koristi bitmapu za reprezentovanje podataka, što ubrzava neke operacije sa objektom. Nedostatak ovog pristupa je da mnoge komercijalne OLE aplikacije na kraju proizvode "pompezne" fajlove (jer se čuvaju dve kopije istih podataka). Ukoliko uzmete u obzir ovaj problem uz relativnu sporost OLE-a i količinu posla koja je potrebna da se programiraju OLE serveri, možete razumeti zašto je upotreba ovog moćnog pristupa još uvek ograničena u poređenju sa onim što je Microsoft predvideo pre nekoliko godina.

Kontejneri složenih dokumenata mogu podržavati OLE u različitom stepenu. Vi možete da smestite objekat u kontejner umetanjem novog objekta, prebacivanjem ili prebacivanjem i povezivanjem sa Clipboarda, prevlačenjem iz druge aplikacije i tako dalje.

Kada se objekat jednom smesti u kontejner, možete nad njim obaviti operacije upotrebom raspoloživih akcija (verbs) servera. Obično je edit verb unapred određena akcija — akcija koja se izvodi kada dva puta kliknete objekat. Za druge objekte, kao što su video i zvučni zapisi, play je akcija koja je unapred određena. Obično možete da vidite spisak akcija kontejner objekta ukoliko ga kliknete desnim tasterom miša. Iste informacije su dostupne iz programa preko elementa menija Edit➥Object, koji ima podmeni koji prikazuje dostupne akcije za taj objekat.

NAPOMENA

Delphi ne obezbeđuje nikakvu vizuelnu podršku za izradu servera složenih dokumenata. Uvek možete da napišete server koji implementira odgovarajuće interfejse. Podrška za kontejner složenog dokumenta se lako dobija pomoću komponente OleContainer. ■

OLE kontejner komponenta

Da biste kreirali jednostavnu OLE kontejner aplikaciju u Delphiju, postavite komponentu OleContainer na formular. Zatim selektujte komponentu i kliknite desnim tasterom miša da biste aktivirali njen lokalni meni koji će sadržati komandu Insert Object. Kada odaberete ovu komandu, Delphi prikazuje standardni okvir za dijalog OLE Insert Object. Ovaj okvir za dijalog Vam omogućava da odaberete jednu od server aplikacija koje su registrovane na kompjuteru.

Kada je OLE objekat umetnut u kontejner, lokalni meni kontrolne kontejner komponente će sadržati još elemenata menija. Novi elementi menija uključuju komande kojima se menjaju svojstva OLE objekta, umeće se jedan objekat, kopira ili uklanja postojeći objekat. Lista, takođe, sadrži akcije objekta (kao što su Edit, Open ili Play). Kada je OLE objekat umetnut u kontejner, odgovarajući server će se pokrenuti da biste mogli da izmenite novi objekat. Čim zatvorite server aplikaciju, Delphi ažurira objekat u kontejneru i prikazuje ga u vreme dizajniranja na formularu Delphi aplikacije koju izrađujete.

Ukoliko pogledate tekstualni opis formulara koji sadrži komponentu (koja u sebi ima objekat), primetićete svojstvo Data koje sadrži aktuelne podatke OLE objekta. Mada klijent program čuva podatke objekta, on ne zna kako da te podatke obradi ili prikaže bez pomoći odgovarajućeg servera (koji mora biti na raspolaganju na kompjuteru na kome izvršavate program). To znači da je OLE objekat ugnežđen.

Da biste u potpunosti podržali složene dokumente, program mora da obezbedi meni i paletu alata ili panel. Ove dodatne komponente su važne jer editovanje na mestu implicira spajanje korisničkog interfejsa klijenta sa interfejsom server programa. Kada se OLE objekat aktivira na mestu, neki od menija linije menija server aplikacije se dodaju liniji menija kontejner aplikacije.

OLE spajanje menija se u Delphiju obavlja gotovo automatski. Potrebno je samo da odredite odgovarajuće indekse za elemente menija kontejnera, upotrebom svojstva GroupIndex. Bilo koji element menija koji ima neparan broj indeksa, zamenjuje se odgovarajućim elementom aktivnog OLE objekta. Preciznije, meniji File (0) i Window (4) pripadaju kontejner aplikaciji. Meniji Edit(1), View (3) i Help (5) (ili grupe menija sa tim indeksima) preuzimaju se sa OLE servera.

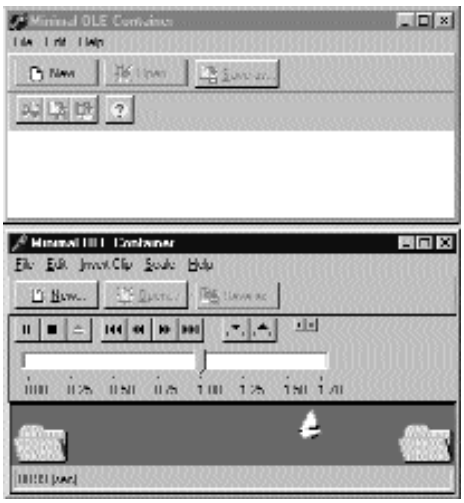
Šestu grupu, nazvanu Object i označenu brojem 2, kontejner može da koristi za prikazivanje još jednog menija između grupa Edit i View, čak i kada je OLE objekat aktivan. Demo program OleCont, koji sam napisao radi demonstracije ovih karakteristika, omogućava korisniku da kreira novi objekat pozivanjem metoda InsertObjectDialog klase TOleContainer.

Metod InsertObjectDialog prikazuje sistemski okvir za dijalog, ali ne aktivira automatski OLE objekat:

```
procedure TForm1.New1Click ( Sender: TObject);
begin
  if OleContainer1.InsertObjectDialog then
    OleContainer1.DoVerb (OleContainer1.PrimaryVerb);
end;
```

Kada je novi objekat kreiran, možete da izvršite njegovu primarnu akciju upotrebom metoda DoVerb. Program prikazuje i malu paletu alata sa nekoliko kontrola sa bitmapama. Ja sam postavio nekoliko komponenata TWinControl na formular da bih korisniku omogućio da ih selektuje i tako onemogući OleContainer. Da bi ova paleta alata/panel bila vidljiva prilikom editovanja na mestu, trebalo bi da odredite vrednost True za svojstvo Locked. Ovim se panel primorava da ostane u aplikaciji i da ga ne zameni paleta alata servera.

Da bih Vam pokazao šta se dešava kada ne koristite ovaj pristup, ja sam programu dodao drugi panel, koji sadrži nekoliko kontrola više. Pošto nisam podesio svojstvo panela Locked, ova nova paleta alata će biti zamenjena paletom alata aktivnog OLE servera. Kada editovanje na mestu pokrene server aplikaciju koja prikazuje paletu alata, paleta alata servera zamenjuje paletu alata kontejnera, kao što možete videti na donjem delu slike 16.11.



SLIKA 16.11 *Druga paleta alata primera OleCont (gore) zamenjena je paletom alata servera (dole)*

SAVET

Da biste učinili da sve automatske operacije menjanja veličine glatko funkcionišu, trebalo bi da postavite OLE kontejner u panel komponentu i da ih oba poravnate sa klijent oblašću formulara. ■

Drugi način za kreiranje OLE objekata je upotreba metoda `PasteSpecialDialog`, koji se poziva u obradi događaja `PasteSpecial1Click` ovog primera. Još jedan standardni OLE okvir za dijalog, koji je obmotan Delphi funkcijom, jeste onaj koji prikazuje svojstva objekta, a koji se aktivira upotrebom elementa Object Properties menija Edit:

```
procedure TForm1.Object1Click (Sender: TObject);
begin
  OleContainer1.ObjectPropertiesDialog;
end;
```

Rezultat standardnog OLE okvira za dijalog možete videti na slici16.12. Očigledno, ovaj okvir za dijalog se menja prema prirodi aktivnog OLE objekta u kontejneru.

Poslednja karakteristika programa OleCont je podrška za fajlove. Ovo je, zapravo, jedan od najlakših dodataka koje smo mogli da načinimo, jer OLE kontejner već obezbeđuje podršku za fajlove.

SLIKA 16.12 *Standardni okvir za dijalog OLE Object Properties, koji je na raspolaganju u primeru OleCont*

Upotreba internih objekata

U prethodnom programu korisnik je određivao tip internog objekta koji kreira program. U ovom slučaju možete učiniti malo toga da biste komunicirali sa internim objektima. Pretpostavimo, umesto toga, da želite da ugnezdite Wordov dokument u Delphi aplikaciju i da ga zatim izmenite upotrebom Delphi koda. Ovo možete učiniti upotrebom OLE Automationa sa ugnežđenim objektom, što je pokazano primerom WordCont (naziv je skraćenica od Word Container — Wordov kontejner).

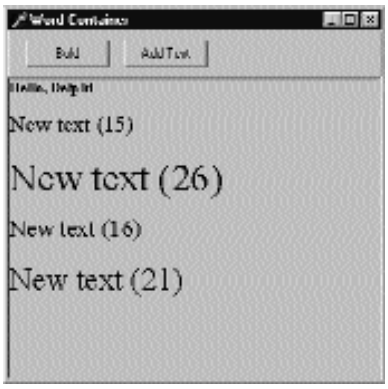
UPOZORENJE

Kako primer WordCont sadrži objekat specifičnog tipa, Microsoft Word dokument, on (primer) se neće izvršavati ukoliko nemate instaliranu tu server aplikaciju. Posedovanje različite verzije servera takođe može da stvori probleme ukoliko Automation metodi servera, koje koristi klijent program, nisu dostupni u Vašoj verziji servera. ■

Ja sam formularu ovog primera dodao komponentu OleContainer, odredio sam vrednost `aaManual` za svojstvo `AutoActive` (tako da je jedina interakcija moguća preko našeg koda) i dodao sam paletu alata sa nekoliko kontrola. Kod dve kontrole je prilično direktan, kada znate da ugnežđeni objekat odgovara Wordovom dokumentu:

```
procedure TForm1.Button1Click (Sender: TObject);
var
  Document: Variant;
begin
  // activates if not running
  if not (OleCantainer1.State = osRunning) then
    OleCantainer1.Run;

  // get the document
  Document := OleCantainer1.OleObject
  // first paragraph to bold
  Document.Paragraphs.Item(1).Range.Bold
end;

procedure TForm1.Button3Click(Sender TObject);
var
  Document, Paragraph: Variant;
begin
  // activate if not running
  if not (OleCantainer1.State = osRunning) then
    OleCantainer1.Run;

  // get the document
  Document := OleCantainer1.OleObject;
  // add paragraphs, getting the last one
  Document.Paragraphs.Add;
  Paragraph : = Document. Paragraphs.Add;
  // add text to the paragraph, using random font size
  Paragraph.Range.Font.Size := 10 + Random (20);
  Paragraph.Range.Text := 'New text (' +
    IntToStr (Paragraph.Range.Font.Size) + ')'#13
end;
```

Efekat ovog koda možete videti na slici 16.13. Kod nije neverovatno moćan, ali pokazuje kako možete da spojite upotrebu tehnika OLE Containers i OLE Automation.



SLIKA 16.13 *Primer WordCont pokazuje kako da koristimo OLE Automation sa ugnežđenim objektom*

Uvod u ActiveX kontrole

Microsoftov Visual Basic je bio prvo programsko razvojno okruženje koje je predstavilo ideju obezbeđivanja softverskih komponenata za veliko tržište. Zapravo, koncept softverskih komponenata koje se mogu ponovo upotrebljavati je stariji od Visual Basica — ima duboke korene u teorijama objektno orijentisanog programiranja (OOP). Međutim, OOP jezici nikada nisu dali ponovnu upotrebu koju su obećali, verovatno više zbog problema marketinga i standardizacije nego zbog bilo čega drugog. Mada Visual Basic ne koristi u potpunosti objektno orijentisano programiranje, on ipak primenjuje koncept komponenata preko svog standardnog načina izrade i distribuiranja novih kontrola koje programeri mogu da integrišu u okruženje.

Prvi tehnički standard koji je promovisao Visual Basic je bio VBX, 16-bitna specifikacija koja je u potpunosti bila na raspolaganju u 16-bitnoj verziji Delphija. Prelaskom na 32-bitne platforme, Microsoft je VBX standard zamenio moćnijim i otvorenijim ActiveX kontrolama.

NAPOMENA

ActiveX kontrole su se nekada zvale OLE Controls (ili OCX). Promena naziva odslikava novu marketinšku strategiju Microsofta pre nego što odslikava tehničke inovacije. Tehnički, ActiveX se može smatrati manjim proširenjem OCX tehnologije. Nije iznenađenje da se ActiveX kontrole obično čuvaju u fajlovima sa ekstenzijom .ocx. ■

Iz opšte perspektive, ActiveX kontrola se ne razlikuje mnogo od Windows, Delphi ili Visual Basic kontrole. Kontrola u bilo kojem od ovoh jezika je uvek prozor, sa pridruženim kodom koji određuje njeno ponašanje. Ključna razlika između različitih porodica kontrola leži u interfejs kontroli, interakciji između kontrole i ostatka aplikacije. Tipične Windows kontrole koriste interfejs koji je zasnovan na porukama, VBX kontrole koriste svojstva i događaje, a ActiveX kontrole koriste svojstva, metode i događaje. Ova tri elementa svojstava, metoda i događaja se mogu naći i u Delphijevim komponentama.

Upotrebom OLE žargona, ActiveX kontrola je "složeni dokument objekat koji je implementiran kao DLL server u procesu i podržava OLE Automation, vizuelno editovanje i aktiviranje unutra-napolje (inside-out) ". Potpuno jasno, zar ne? Hajde da vidimo šta definicija zapravo znači.

ActiveX kontrola koristi isti pristup kao i OLE server objekat, a to su objekti koje možete da umetnete u OLE Document, kao što smo videli u prethodnom poglavlju. Razlika između generičkog OLE servera i ActiveX kontrole je u tome da ActiveX kontrole mogu biti implementirane na jedan način, dok OLE serveri mogu biti implementirani na tri različita načina:

- kao samostalne aplikacije (na primer, Microsoft Excel);
- kao serveri van procesa — to jest, izvršni fajlovi koji se ne mogu samostalno izvršavati, a može ih pozvati samo server (na primer, Microsoft Graph i slične aplikacije);
- kao serveri u procesu, kakvi su DLL-ovi koji se učitavaju u isti memorijski prostor kao i program koji ih koristi.

ActiveX kontrole mogu biti implementirane samo upotrebom poslednje tehnike, koja je, između ostalog, i najbrža: dakle, kao serveri u procesu. ActiveX kontrole su OLE Automation serveri (to smo razmatrali u prethodnom poglavlju). To znači da možete da pristupate svojstvima ovih objekata i pozivate njihove metode.

ActiveX kontrolu možete videti u aplikaciji koja je koristi i sa njom možete da imate direktnu interakciju u kontejner prozoru aplikacije. To je značenje termina vizuelno editovanje (visual editing), ili aktiviranje na mestu (in-place activation). Jednim klikom možete da aktivirate kontrolu umesto da koristite dva klika kao kod OLE Documents, a kontrola je aktivna kad god je vidljiva (to je ono što znači termin aktiviranje unutra-spolja), a da ne morate da je kliknete dva puta.

Kao što sam ranije pomenuo, ActiveX kontrola sadrži svojstva, metode i događaje. Svojstva mogu da označavaju status, ali mogu i da aktiviraju metode. (Ovo naročito važi za ActiveX kontrole koje predstavljaju unapređene VBX kontrole, jer kod VBX kontrola nije bilo drugog načna da aktivirate metod nego da podesite svojstvo.) Svojstva mogu da se referišu na agregatne vrednosti, nizove, podobjekte i tako dalje. Svojstva takođe mogu biti dinamička (ili, rečeno Delphi terminologijom, samo za čitanje).

U ActiveX kontroli svojstva su podeljena u različite grupe: standardna svojstva koja većina kontrola mora da implementira; svojstva koja nude informacije o kontejneru (slično svojstvima ParentColor ili ParentFont u Delphiju); proširena svojstva kojima upravlja kontejner, kao što je pozicija objekta; korisnička svojstva, a to može da bude bilo šta.

Događaji i metodi su događaji i metodi. Događaji se odnose na klik mišem, pritisak na taster, aktiviranje komponente i druge specifične akcije korisnika. Metodi su funkcije i procedure koje se odnose na kontrolu. Ne postoji velika razlika između ActiveX i Delphi koncepata događaja i metoda.

ActiveX kontrole nasuprot Delphi kontrolama

Pre nego što Vam pokažem kako da napišete i koristite ActiveX kontrole u Delphiju, upoznajmo se sa nekim tehničkim razlikama između ove dve vrste kontrola. ActiveX kontrole su zasnovane na DLL-ovima. To znači da je, kada ih koristite, potrebno da distribuirate njihov kod (OCX fajl) uz aplikaciju koja ih koristi. U Delphiju kod komponenata može biti statički povezan sa izvršnim fajlom, ili može biti dinamički povezan upotrebom paketa samo za vreme izvršavanja, te uvek možete da odaberete način povezivanja.

Postojanje zasebnog fajla Vam omogućava da kod delite između aplikacija, što DLL-ovi obično čine. Ukoliko dve aplikacije koriste istu kontrolu (ili paket samo za vreme izvršavanja), potrebna Vam je samo jedna kopija na hard disku i samo jedna kopija u memoriji. Nedostatak je da ukoliko dva programa moraju da koriste dve različite verzije ActiveX kontrole, mogu se javiti neki problemi kompatibilnosti. Prednost postojanja izvršnog fajla koji sadrži sam sebe je u tome što ćete imati manje problema prilikom instalacije.

Sada, kakav je nedostatak upotrebe Delphi komponenata? Pravi problem nije to što postoji manje Delphi komponenata od ActiveX kontrola, već što ukoliko kupite Delphi komponentu, moći ćete da je koristite samo u Delphiju i Borland C++ Builderu. Ukoliko, s druge strane, kupite ActiveX kontrolu, moći ćete da je koristite u različitim razvojnim okruženjima različitih proizvođača. I pored svega, ukoliko uglavnom programirate u Delphiju i pronađete dve slične komponente koje se zasnivaju na dvema tehnologijama, ja Vam predlažem da kupite Delphi komponentu — biće više integrisana sa Vašim okruženjem i zbog toga će njena upotreba biti bezbednija. Takođe, Delphi komponente će, verovatno, biti bolje dokumentovane (iz Pascal perspektive), iskoristiće Delphi i Object Pascal karakteristike koje nemate u opštem ActiveX interfejsu, koji se tradicionalno zasniva na jezicima C i C++.

Upotreba ActiveX kontrola u Delphiju

Delphi dobijate sa nekoliko unapred obezbeđenih ActiveX kontrola, a možete veoma lako da kupite i instalirate i druge ActiveX kontrole. Posle ovog opisa funkcionisanja ActiveX kontrola, jednu ću upotrebiti u jednostavnom primeru.

Delphijev proces instalacije je veoma jednostavan. Odaberite Component➥Import ActiveX Control iz Delphi menija. Na ovaj način se otvara okvir za dijalog ActiveX u kome možete videti spisak biblioteka ActiveX kontrola koje su registrovane u Windowsu. Ukoliko odaberete jednu, Delphi će pročitati njen Type Library, prikazati kontrole i predložiti naziv fajla za njenu jedinicu. Ukoliko su informacije korektne, jednostavno kliknite kontrolu Create Unit da biste pogledali Pascalov fajl sa izvornim kodom kreiran u Delphiju kao omotač za ActiveX kontrolu. Kliknite kontrolu Install da biste ovu novu jedinicu dodali Delphi paketu i u Component Palette.

Upotreba kontrole WebBrowser

Da bih izradio primer, ja sam koristio unapred obezbeđene ActiveX kontrole koje su na raspolaganju u Delphiju. Za razliku od kontrola nezavisnih programera, ovo nije moguće učiniti na ActiveX strani palete već na strani Internet. Kontrola, nazvana WebBrowser, je samo omotač oko Microsoftovog mehanizma Internet Explorer. Primer predstavlja veoma jednostavan web pretraživač.

Program WebBrowser sadrži ActiveX kontrolu TWebBrowser, koja prekriva klijent oblast i kontrolnu liniju na vrhu i statusnu liniju na dnu. Da biste prešli na neku web stranu, potrebno je da izvršite unos u combo polje palete alata, odaberete neku od posećenih URL adresa (koje se čuvaju u combo polju) ili kliknete kontrolu Open File da biste otvorili lokalni fajl.

Implementacija koda koji se koristi za selektovanje web ili lokalnog HTML fajla, nalazi se u metodu GotoPage:

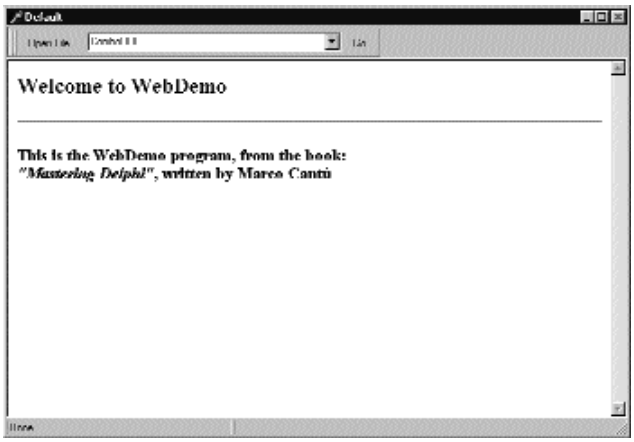
```
procedure TForm1.GotoPage (ReqUrl: string);
begin
  WebBrowser1.Navigate (ReqUrl, EmptyParam, EmptyParam,
    EmptyParam, EmptyParam);
end;
```

EmptyParam je unapred određeni OleVariant koji možete da koristite svaki put kada želite da prosledite unapred određenu vrednost kao parametar reference. Ovo je zgodna prečica koju možete koristiti da biste izbegli kreiranje praznog OleVarianta svaki put kada Vam je potreban sličan parametar. Ovaj metod se poziva za fajl, kada korisnik klikne kontrolu Enter u combo polju, ili kada odabere kontrolu Go:

```
procedure TForm1.ComboURLKeyClick (Sender: TObject; var Key: Char);
begin
  if OpenDialof1.Execute then
    GotoPage (OpenDialog1.FileName);
end;

procedure TForm1.ComboURLKeyPress (Sender: TObject; var Key: Char);
begin
  if Key = #13 then
    GotoPage (ComboUrl.Text);
end;

procedure TForm1.BtnGoClick (Sender: TObject);
begin
  GotoPage (ComboUrl.Text);
end;
```

Zapravo, postoji i četvrta upotreba metoda `GotoPage`. Kada se program pokrene, učitava se pozdravni HTML iz trenutnog direktorijuma, a efekat možete da vidite na slici 16.14:

```
procedure TForm1.FormShow (Sender: TObject);
begin
  GotoPage (ExtractFilePath (Application.ExeName) +
    'greeting.htm');
end;
```



SLIKA 16.14 *Program WebDemo na početku izvršavanja. Kada ga budete koristili, videćete da u potpunosti podržava grafiku i druge web ekstenzije, jer je zasnovan na Microsoft mehanizmu Internet Explorer*

Program, takođe, obrađuje četiri događaja kontrole WebBrowser. Kada započne i završi se operacija preuzimanja, program ažurira tekst statusne linije kao i listu combo polja:

```
procedure TForm1.WebBrowser1DownlaodBegin (Sender: TObject);
begin
  StatusBar1.Panels[0].Text := 'Downloading ' +
    WebBrowser1.LocationURL + '. . .';
end;

procedure TForm1.WebBrowser1DownloadComplete (Sender: TObject);
var
  NewUrl: string;
begin
  StatusBar1.Panels[0].Text := 'Done';
  // add URL to combobox
  NewUrl := WebBrowser1.LocationURL;
  if (NewUrl <> '') and
    (ComboURL.Items.IndexOf (NewUrl) < 0) then
  ComboURL.Items.Add (NewUrl);
end;
```

Druga dva korisna događaja su događaj OnTitleChange, koji se koristi za ažuriranje zaglavlja naslovom HTML dokumenta, i OnStatusTextChange, koji se koristi za ažuriranje drugog dela statusne linije. Ovaj kod je u osnovi dupliran u prvom delu statusne linije prethodne dve obrade događaja:

```
procedure TForm1.WebBrowser1TitleChange (Sender: TObject;
  const Text: WideString);
begin
  Caption := Text;
end;
```

```
procedure TForm1.WebBrowser1StatusTextChange (Sender: TObject;
  const Text: WideString);
begin
  statusBar1.Panels[1].Text := Text;
end;
```

Pisanje ActiveX kontrola

Pored upotrebe postojećih, u Delphiju lako možete da načinite nove ActiveX kontrole. Mada sami možete da napišete kod nove ActiveX kontrole, implementiranje svih potrebnih ActiveX interfejsa (a ima ih mnogo) je mnogo lakše upotrebom jedne od tehnika koje su direktno podržane u Delphiju:

- Možete da upotrebite ActiveX Control Wizard da biste VCL kontrolu pretvorili u ActiveX kontrolu. Počinjete od postojeće VCL komponente, koja mora biti naslednik komponente `TWinControl`, a Delphi omotava ActiveX oko nje. Tokom ovog koraka Delphi kontroli dodaje Type Library. (Omotavanje ActiveX kontrole oko Delphi komponente je upravo suprotno od onoga što smo činili da bismo koristili ActiveX kontrolu u Delphiju.)
- Možete kreirati ActiveForm, na njega smestiti nekoliko kontrola i proslediti ceo formular (bez bordura) kao ActiveX kontrolu. Ova druga tehnika je ista kao tehnika koju koristi Visual Basic i generalno je namenjena izradi Internet aplikacija. Ipak, to je veoma dobra alternativa za konstrukciju ActiveX kontrole zasnovane na više Delphi kontrola ili Delphi komponenata koje nisu naslednici `TWinControl` klase.

Opcioni korak koji možete preduzeti u oba slučaja je da pripremite stranu sa svojstvima za kontrolu, da je upotrebite kao vrstu editora svojstava za određivanje početnih vrednosti svojstava kontrole u razvojnom okruženju. To je neka vrsta alternative Object Inspectora iz Delphija. Kako većina razvojnih okruženja omogućava samo ograničeno editovanje, mnogo je važnije napisati stranu sa svojstvima nego napisati komponentu ili editor svojstva za Delphi kontrolu.

Izrada ActiveX strelice

Za primer programiranja ActiveX kontrole odlučio sam da uzmem komponentu Arrow koju smo načinili u Poglavlju 13 i pretvorim je u ActiveX. Zapravo, mi komponentu ne možemo direktno da upotrebimo, jer je to grafička kontrola, potklasa klase `TGraphicControl`. Ipak, pretvaranje grafičke kontrole u kontrolu koja se nalazi u prozoru je obično direktna operacija.

U ovom slučaju sam samo promenio naziv osnovne klase u `TCustomControl` (a takođe sam promenio naziv klase kontrole da bih izbegao koliziju naziva):

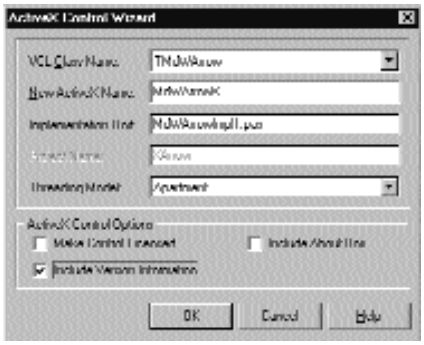
```
type
  TMdWArrow = class (TCustomControl)
  ...
```

Klasa `TWinControl` ima minimalnu podršku grafičkog izlaza. Njena potklasa `TCustomControl` u osnovi ima iste mogućnosti kao i klasa `TGraphicControl`. Suštinska razlika je u tome da `TCustomControl` sadrži hendl prozora.

Posle instaliranja ove nove komponente u Delphiju, spremni smo da programiramo novi primer. Da biste kreirali novu ActiveX biblioteku, jednostavno odaberite File➥New, pređite na stranu ActiveX i odaberite ActiveX biblioteku. Delphi kreira praznu strukturu DLL-a, kao što smo videli na početku ovog poglavlja. Ja sam ovu biblioteku sačuvao pod imenom XArrow, u direktorijumu sa istim nazivom, kao i obično.

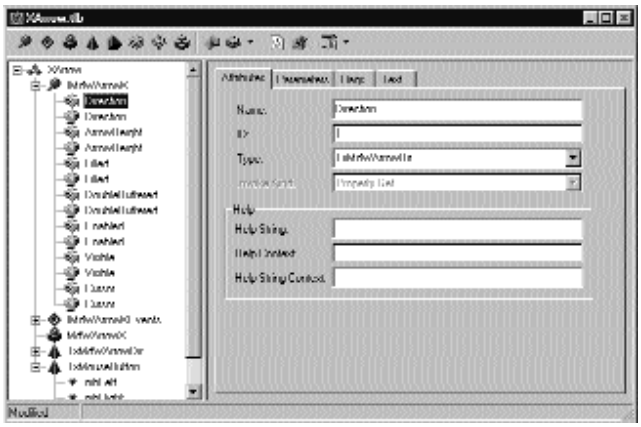
Sada je vreme da upotrebimo ActiveX Control Wizard, koji je na raspolaganju na ActiveX strani Object Repositoryja — Delphijevom okviru za dijalog New. U ovom čarobnjaku (koji je prikazan na slici 16.15) birajte VCL klasu za koju ste zainteresovani, prilagodite nazive koji su prikazani u poljima za izmene i kliknite OK; Delphi zatim obezbeđuje kompletan kod ActiveX kontrole za Vas.

Upotreba tri polja za potvrdu na dnu prozora ActiveX Control Wizarda možda nije očigledna. Ukoliko uključite licencu podrške za vreme dizajniranja, korisnik kontrole neće moći da koristi kontrolu u razvojnom okruženju ako ne poseduje odgovarajuću licencu kontrole (license key). Drugo polje za potvrdu Vam omogućava da uključite informacije o verziji ActiveX kontrole u OCX fajl. (Informacije o verziji se razmatraju u Poglavlju 19.) Ukoliko je potvrđeno treće polje, ActiveX Control Wizard kontroli automatski dodaje polje About.



SLIKA 16.15 *Delphijev ActiveX Control Wizard*

Pogledajmo kod koji generiše ActiveX Control Wizard. Ključni element ovog čarobnjaka je generisanje biblioteke Type Library. Biblioteku, koja je generisana za našu kontrolu strelice, možete videti u Delphijevom editoru Type Library koji je prikazan na slici 16.16. Na osnovu informacija iz Type Libraryja čarobnjak takođe generiše uvozni fajl sa definicijom interfejsa, dispinterface i druge tipove i konstante.



SLIKA 16.16 *Editor Type Library koji prikazuje Type Library ActiveX kontrole koju sam ja kreirao*

U ovom primeru uvozni fajl je nazvan `XArrow_TLB.PAS`. Prvi deo ovog fajla sadrži nekoliko GUID-ova, jedan za celu biblioteku i jedan za kontrolu i druge konstante za definiciju odgovarajućih OLE pobrojanih tipova podataka koji koriste svojstva Delphi kontrole, na primer:

```
type
  TxMdWArrowDir = TOleEnum;

const
  adUp = $00000000;
  adLeft = $00000001;
  adDown = $00000002;
  adRight = $00000003;
```

Suština je deklaracija interfejsa `IMdWArrowX`, koju Vam preporučujem da pogledate u izvornom kodu. Primetićete da poslednji deo uvozne jedinice uključuje deklaraciju klase `TMdWArrowX`. To je klasa izvedena iz klase `TOleControl` koju možete upotrebiti za instaliranje kontrole u Delphiju, kao što smo videli u prvom delu ovog poglavlja. Ova klasa Vam nije neophodna za izradu ActiveX kontrole. Potrebna Vam je samo za instaliranje ActiveX kontrole u Delphiju. Klasa koju koristi ActiveX server ima isti naziv, ali drugačiju implementaciju.

Ostatak koda, i kod koji ćete prilagoditi, nalazi se u glavnoj jedinici, koju sam ja u ovom primeru nazvao `MdWArrowImpl1`. Ova jedinica sadrži deklaraciju ActiveX server objekta, koji je izveden iz klase TActiveXControl i implementira specifični interfejs `IMdWArrowX`:

```
type
  TMdWArrowX = class (TActiveXControl, IMdWArrowX)
  ...
```

NAPOMENA

Klasa TActiveXControl obavlja veći deo posla za obezebđivanje ActiveX podrške u Delphiju. Ova klasa implementira brojne interfejse koji su neophodni za svaku ActiveX kontrolu: `IConnectionPointContainer`, `IDataObject`, `IObjectSafety`, `IOleControl`, `IOleInPlaceActiveObject`, `IOleInPlaceObject`, `IOleObject`, `IPerPropertyBrowsing`, `IPersistPropertyBag`, `IPersistStorage`, `IPersistStreamInit`, `IQuickActivate`, `ISimpleFrameSite`, `ISpecifyPropertyPages`, `IViewObject` i `IViewObject2`. Samo deklaracija klase `TActiveXControl` je duga više od 250 linija koda, a njena implementacija je odgovorna za dobar deo od 4000 linija koda `AxCtrls` jedinice. ■

Pre nego što ovu kontrolu prilagodimo na bilo koji način, pogledajmo kako funkcioniše. Prvo bi trebalo da kompajlirate ActiveX biblioteku i da je zatim registrujete upotrebom Delphijeve komande menija Run➥Register ActiveX Server. Sada možete da instalirate ActiveX kontrolu kao što smo to ranije učinili, izuzev što morate da navedete drugi naziv za novu klasu da biste izbegli konflikt. Ukoliko upotrebite kontrolu, ona ne izgleda mnogo drugačije od originalne VCL kontrole, ali prednost je to što ista komponenta sada može da se instalira i u druga razvojna okruženja.

Dodavanje novih svojstava

Kada ste kreirali ActiveX kontrolu, dodavanje novih svojstava, događaja ili metoda kontroli — na veliko iznenađenje —lakše je od iste operacije za VCL komponente. Delphi, zapravo, obezbeđuje specifičnu vizuelnu podršku za prvu operaciju, ali ne i za drugu.

Jednostavno možete da otvorite Pascal jedinicu sa implementacijom ActiveX kontrole i odaberete Edit➥Add To Interface. Drugi način je da istu komandu odaberete iz lokalnog menija editora. Delphi otvara okvir za dijalog Add to Interface (videti sliku 16.17). U combo polju okvira za dijalog možete da odaberete novo svojstvo, metod ili događaj. U ovom primeru, prvi izbor će uticati na interfejs `IMdWArrowX`, a drugi na interfejs `IMdWArrowXEvents`.

SLIKA 16.17 *Okvir za dijalog Add to Interface kada je aktivna pomoć za sintaksu*

U polje za izmene možete da unesete deklaraciju novog elementa interfejsa. Ukoliko je aktiviran Syntax Helper, dobijaćete oblačiće u kojima je opis šta treba sledeće da unesete, a greške će biti istaknute. Sintaksnu pomoć u akciji vidite na slici 16.17. Kada definišete novi ActiveX interfejs element, imajte na umu da ste ograničeni na OLE tipove podataka. U primeru XArrow ja sam na raspolaganje stavio boju. Ovo su primeri onoga što možete napisati u polju za izmene okvira Add to Interface (koji se dva puta izvršava):

```
property FillColor: Integer;
property PenColor: Integer;
```

NAPOMENA

Pošto je `TColor` specifična Delphi definicija, nije dozvoljeno da je koristite. TColor je podskup celih brojeva kojem je unapred određena veličina celih brojeva, te sam upotrebio standardni Integer tip podataka. ■

Deklaracije koje unosite u okvir za dijalog Add to Interface automatski se dodaju Type Library (TLB) fajlu kontrole, njenoj uvoznoj jedinici i njenoj implementacionoj jedinici:

```
type
  IMdWArrowX = interface (IDispatch)
    function Get_FillColor: Integer; safecall;
    procedure Set_ FillColor (Value: Integer); safecall;
    function Get_PenColor: Integer; safeceall;
    procedure Set_PenColor (Value: Integer); safecall;
    ...
    property FillColor: Integer
      read Get_ FillColor write Set_ FillColor;
    property PenColor: Integer
      read Get_PenColor write Set_PenColor;
```

Sve što je potrebno da učinite da biste dovršili ActiveX kontrolu jeste da popunite metode implementacije Get i Set. Evo koda prvog svojstva:

```
function TMdWArrowX.Get_ FillColor: Integer;
begin
  Rsult := ColorToRGB (FDelphiControl.Brush.Color);
end;

procedure TMdWArrowX.Set_ FillColor (Value: Integer);
begin
  FDelphiControl.Brush.Color := Value;
end;
```

Ukoliko sada još jednom instalirate ovu ActiveX kontrolu u Delphi okruženje, prikazaće se dva nova svojstva. Jedini problem sa ovim svojstvom je što Delphi koristi običan editor za cele brojeve, što čini prilično komplikovanim unošenje nove vrednosti boje. Program, nasuprot tome, lako može da koristi `RGB` funkciju za kreiranje odgovarajuće vrednosti boje.

Dodavanje strane svojstva

Činjenica je da druga razvojna okruženja mogu malo toga učiniti sa našom komponentom jer nismo pripremili nikakvu stranu za svojstva — nikakav editor svojstva. Strana svojstva je osnovna stvar tako da programeri koji koriste komponentu mogu da promene njene atribute. Ipak, dodavanje strane svojstva nije jednostavno koliko dodavanje formulara sa nekoliko kontrola. Strana svojstva će se, zapravo, integrisati sa razvojnim okruženjem u kojem radite. Strana svojstva za našu kontrolu će se prikazati unutar strane svojstva okvira za dijalog okruženja, koji će obezbediti kontrole OK, Cancel i Apply, i kartice za prikazivanje više strana svojstava (od kojih neke mogu obezbediti razvojno okruženje u kojem radite). Lepo je što je podrška za strane svojstava ugrađena u Delphi te je dodavanje ovakve strane prilično jednostavno. Jednostavno otvorite ActiveX projekat, zatim kao i obično otvorite okvir za dijalog New Items, pređite na stranu ActiveX i odaberite stranu Property. Ono što ćete dobiti ne razlikuje se mnogo od formulara. Zapravo, klasa `TPropertyPage1` (koja se po definiciji kreira) izvedena je iz klase `TPropertyPage` VCL-a, koja je izvedena iz klase `TCustomForm`.

SAVET

Delphi obezbeđuje četiri ugrađene strane svojstava za boje, fontove, slike i stringove. GUID-ovi ovih klasa su naznačeni konstantama `Class_DColorPropPage`, `Class_DFontPropPage`, `Class_DPicturePropPage` i `Class_DStringPropPage` u jedinici `AxCtrls`. ■

Na strani svojstva možete dodati kontrole kao što biste to učinili sa Delphi formularom i možete napisati kod preko koga kontrole međusobno komuniciraju. Ja sam strani svojstva dodao combo polje sa mogućim vrednostima svojstva `Direction`, polje za potvrdu za svojstvo `Filled`, polje za izmene za kontrolu UpDown da bih podesio svojstvo `ArrowHeight`, i dva oblika sa odgovarajućim kontrolama za boje. Jedini kod koji je dodat formularu odnosi se na dve kontrole koje se koriste za promenu boje dveju komponenata, koje omogućavaju da prvo vidite boju aktuelne ActiveX kontrole. Događaj `OnClick` kontrole koristi komponentu ColorDialog, kao i obično:

```
procedure TPropertyPage1.ButtonPenClick (Sender: TObject);
begin
  with ColorDialog1 do
  begin
    Color := ShapePen.Brush.Color;
    if Execute then
    begin
      ShapePen.Brush.Color ;= Color;
      Modified; // enable Apply button!
    end;
  end;
end;
```

Ono što je važno primetiti u ovom kodu, jeste poziv metoda `Modified` klase `TPropertyPage`. Ovaj poziv je neophodan da bi okvir za dijalog strane svojstva znao kada smo izmenili jednu od vrednosti i da bi aktivirao kontrolu Apply. Kada korisnik komunicira sa nekom od kontrola formulara, ovaj poziv se automatski obavlja. Ovu liniju moramo sami dodati za dve kontrole.

SAVET

Drugi savet se odnosi na stranu `Caption` svojstva formulara. Ona će se koristiti u okviru za dijalog svojstva okruženja kao zaglavlje kartice koja odgovara strani svojstva. ■

Sledeći korak je pridruživanje kontrola strane svojstva stvarnim svojstvima ActiveX kontrole. Klasa strane svojstva automatski sadrži dva metoda: `UpdateOleObject` i `UpdatePropertyPage`. Kao što nazivi nagoveštavaju, ovi metodi kopiraju podatke sa strane svojstva u ActiveX kontrolu i obrnuto. Evo koda za moj primer:

```
procedure TPropertyPage1.UpdatePropertyPage;
begin
  { Update your controls from the OleObject }
  ComboDir.ItemIndex := OleObject.Direction;
  CheckFilled.Checked := OleObject.Filled;
  EditHeight.Text := IntToStr (OleObject.ArrowHeight);
  ShapePen.Brush.Color := OleObject.PenColor;
  ShapePoint.Brush.Color := OleObject.FillColor;
end;

procedure TPropertyPage1.UpdateObject;
```

```
begin
  { Update the OleObject from your controls }
  OleObject.Direction := ComboDir.ItemIndex;
  OleObject.Filled := CheckFilled.Checked;
  OleObject.ArrowHeight := UpDownHeight.Position;
  OleObject.PenColor:= ColorToRGB (ShapePen.Brush.Color);
  OleObject.FillColor:=ColorToRGB (ShapePoint.Brush.Color);
end;
```

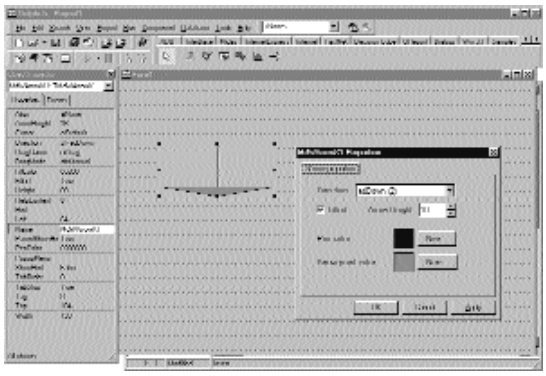
Poslednji korak je povezivanje strane svojstva sa ActiveX kontrolom. Kada je kontrola bila kreirana, Delphi ActiveX Control Wizard je automatski dodao deklaraciju metoda `DefinePropertyPages` implementacionoj jedinici. U ovom metodu ćemo jednostavno pozvati metod `DefinePropertyPage` (ovoga puta je naziv metoda u jednini) za svaku stranu svojstva koju želimo da dodamo ActiveX kontroli. Ovaj metod kao svoj prametar sadrži GUID strane svojstva, nešto što možete pronaći u odgovarajućoj jedinici. (Naravno, potrebno je da dodate uses iskaze koji se referišu na tu jedinicu.) Evo koda mog primera:

```
procedure TMdWArrowX.DefinePropertyPages (
  DefinePropertyPage: TDefinePropertyPage);
begin
  DefinePropertyPage (Class_PropertyPage1);
end;
```

NAPOMENA

Veza između ActiveX kontrole i njene strane svojstva se obavlja preko GUID-a. Ovo je moguće jer objekat strane svojstva može da bude kreiran preko radionice klase, a njen GUID se čuva u Windows Registryju kada registrujete ActiveX biblioteku kontrole. Da biste videli šta se dešava, pogledajte odeljak initialization strane svojstva jedinice, koji poziva TActiveXPropertyPageFactory.Create. ■

Sada kada smo završili programiranje strane svojstva, kada smo ponovo kompajlirali i ponovo registrovali ActiveX biblioteku, možemo da instaliramo ActiveX kontrolu unutar razvojnog okruženja u kome radimo (uključujući i sam Delphi) i vidimo kako izgleda. Na slici 16.18 je prikazan primer. (Ukoliko ste već instalirali ActiveX kontrolu u Delphi, morate je prvo deinstalirati da biste je ponovo izradili. Ovaj proces može zahtevati da Delphi zatvorite i ponovo otvorite.)



SLIKA 16.18 *ActiveX kontrola XArrow i njena strana svojstva u Delphi okruženju*

ActiveForms

Kao što sam ranije pomenuo, Delphi obezbeđuje alternativu za upotrebu ActiveX Control Wizarda za generisanje ActiveX kontrole. Možete upotrebiti ActiveForm, koji je ActiveX kontrola bazirana na formularu, a koja može da sadrži jednu ili više Delphi kontrola. To je upravo tehnika koja se koristi u Visual Basicu za izradu novih kontrola, i ima smisla koristiti je kada kreirate složeni dokument.

Na primer, da biste kreirali ActiveX časovnik, možete jednostavno na ActiveForm smestiti oznaku (koja je grafička kontrola koja se ne može upotrebiti kao polazna tačka za ActiveX kontrolu) i tajmer, i povezati ih upotrebom koda. Formular/kontrola u osnovi postaje kontejner za druge kontrole, što izradu složenih dokumenata čini veoma lakom (izrada je lakša od izrade VCL složenih dokumenata).

Da biste izradili ovakvu kontrolu, jednostavno zatvorite aktuelni projekat i odaberite ikonu ActiveForm sa ActiveX strane okvira za dijalog File➥New. Delphi će od Vas zatražiti neke informacije u narednom okviru za dijalog ActiveForm Wizard, koji je sličan okviru za dijalog ActiveX Control Wizard.

Unutrašnjost ActiveForma

Pre nego što nastavimo primer, pogledajmo kod koji generiše ActiveForm Wizard. Suštinska razlika između običnog Delphi formulara i ActiveForm formulara je u deklaraciji nove klase formulara, koja je izvedena iz klase TActiveForm i implementira specifični ActiveForm interfejs:

```
type
  TAXForm1 = class (TActiveForm, IAXForm1)
```

Kao i obično, interfejs IAXForm je deklarisan u Type Libraryju i u odgovarajućem Pascal fajlu koji generiše Delphi. Evo malog dela interfejsa `IAXFrom1` izdvojenog iz fajla `XF1Lib.pas`, kojem sam ja dodao neke komentare:

```
type
  IAXForm1 = interface (IDispatch)
    ['{51661AA16-9468-11D0-98D0-444553540000}']
    // Get i Set methods for TForm properties
    function Get_Caption: WideString; safecall;
    procedure Set_Caption (const Value: WideString); safecall;
    ...
    // TForm methods redeclared
    procedure Close; safecall;
    ...
    // TForm properties
     property Caption: WideString
       read SetCaption write Get_Caption;
```

Kod generisan za klasu TAXForm1 implementira sve metode `Set` i `Get`, koji jednostavno menjaju ili daju odgovarajuća svojstva formulara, i implementira događaje, koji su opet događaji formulara. Evo malog isečka iz koda:

```
private
  procedure ActivateEvent (Sender: TObject);
protected
  procedure Initialize; override;
  function Get_Caption: WideString; safecall;
  procedure Close; safecall;
  procedure Set_Caption (const Value: WdeString); safecall;
```

Prvo pogledajmo implementaciju svojstava:

```
function TAXForm1.Get_Caption: WideString;
begin
  Result := WideString(Caption);
end;

procedure TAXForm1.Set_Caption (const Value: WideString);
begin
  Caption := TCaption (Value);
  OnActivate := ActivateEvent;
  ...
end;
```

Događaji TForm su podešeni na interne metode kada se formular kreira:

```
procedure TAXForm1.Initialize;
begin
  OnActivate := ActivateEvent;
  ...
end;
```

Svaki događaj zatim mapira samog sebe u ActiveX događaj, kao što je slučaj sa dva naredna metoda:

```
procedure TAXForm1.ActivateEvent (Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnActivate'
end;
```

Upravo zbog ovog mapiranja ne bi trebalo da direktno obrađujete događaje. Umesto toga možete da dodate kod ovim unapred određenim obradama ili možete da zaobiđete metode `TForm` koji pozivaju događaje. (Ovo je upravo pristup koji koristite kada izrađujete Delphi komponente.) Imajte na umu da su svojstva interfejsa ActiveForma namenjena programerima koji koriste komponentu, a ne krajnjim korisnicima ActiveForma na Webu. Ovaj problem mapiranja se odnosi samo na događaje samog formulara, ne na događaje komponenata koje se nalaze na formularu. Možete i dalje da obrađujete događaje komponenata na uobičajen način.

ActiveX kontrola XClock

Kada smo proučili kod koji je Delphi generisao, možemo da se vratimo na programiranje našeg primera XClock. Jednostavno smestite komponente na formualr i podesite njihova svojstva na sledeći način:

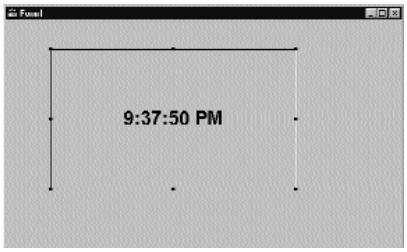
```
object XClock: TXClock
  AxBorderStyle = afbSunken
  Caption = 'XClock'
  Color = clBtnFace
  object Label1: TLabel
    Align = alClient
    Alignment = taCenter
    Font.Height = -27
    FontName = 'Arial'
    FontStyle = [fsBold]
    Layout = tlCenterend
  end
  object Timer1: TTimer
    OnTimer = Timer1Timer
  end
end
```

Poslednji korak je pisanje obrade događaja za događaj `OnTimer` samog tajmera, tako da kontrola ažurira izlaz oznake trenutnim vremenom svake sekunde:

```
procedure TXClock.Timer1Timer(Sender: TObject);
begin
  Label1.Caption := TimeToStr (Time);
end;
```

Sada jednostavno kompajlirajte ovu biblioteku, registrujte je i instalirajte je u paket da biste je testirali u Delphi okruženju. Primer upotrebe komponente možete videti na slici 16.19. Na slici ćete primetiti efekat sunken bordure. Ovo se kontroliše svojstvom **`AxBorderStyle`** aktivnog formulara, koje je jedno od nekoliko svojstava aktivnih formulara koje nije na raspolaganju za obične formulare.

ActiveForms se obično smatraju tehnikama za prosleđvanje Delphi aplikacija preko Interneta. Ipak, ActiveX i ActiveForm podrška koju obezbeđuje Delphi predstavlja različite načine za izradu ActiveX kontrola, koje se mogu koristiti kako na web strani tako i u nekom drugom razvojnom okruženju.



SLIKA 16.19 *ActiveX tajmer instaliran u Delphi paketu*

Šta je sledeće?

U ovom poglavlju sam razmatrao aplikacije Microsoftove COM tehnologije, objašnjavajući Automation, Documents i Controls. Videli smo kako Delphi čini veoma jednostavnim programiranje Automation servera i klijenata i ActiveX kontrola. Delphi 5 je poboljšao ovu podršku omogućavajući nam da omotamo jednostavne komponente oko Automation servera, kakvi su Word ili Excel.

Na druge elemente koji se odnose na COM ćemo se vratiti kada budemo razmatrali Internet i distribuirane aplikacije u poglavljima 20 i 21. Za sada, u Delu V ćemo obratiti više pažnje na druge interesantne programerske zadatke, kao što su više procesa (multithreading), debagovanje (otklanjanje grešaka) i štampanje. Naredna poglavlja Vam nude neke sastavne blokove za programiranje praktičnih (real-world) aplikacija. Drugim rečima, naredna poglavlja pokušavaju da daju odgovor na uobičajene svakodnevne progrmerske probleme.

DEO

V

Praktične tehnike

U OVOM DELU:

POGLAVLJE

17

Multitasking, više procesa i sinhronizacija

Platforma Windows 32 omogućava simultano izvršavanje mnogih programa i aktiviranje više konkurentnih procesa izvršavanja jednog programa. Mada postoje razlike između operativnih sistema Windows NT (sada Windows 2000) i Windows 95/98, većina ključnih elemenata je zajednička za celu platformu Win32.

U ovom poglavlju ćemo razmatrati procese, mutekse i objekte sinhronizacije. Takođe ćemo se ponovo baviti porukama i nekim `Application` događajima, koji u nekim slučajevima nude jednostavnije rešenje za potrebe izvršavanja programa u pozadini.

Događaji, poruke i multitasking u Windowsu

Da bismo razumeli kako Windows aplikacije funkcionišu interno, potrebno je da ukratko razmotrimo kako je multitasking podržan u ovom okruženju. Takođe je potrebno da shvatimo ulogu tajmera (i komponente Timer) i izvršavanja u pozadini (ili *idle* — kada aplikacija nema posla).

Ukratko, moramo se dublje uneti u strukturu Windowsa vođenu događajima i njegovu multitasking podršku. Kako je ovo knjiga o *Delphi* programiranju, ja neću detaljno razmatrati ove teme, ali ću dati pregled zbog čitalaca koji imaju ograničeno iskustvo sa Windows API programiranjem.

Programiranje vođeno događajima

Osnovna ideja koja stoji iza programiranja vođenog događajima je ta da specifični događaji kontrolišu tok aplikacije. Program veći deo svog vremena provede čekajući ove događaje i obezbeđuje kod kojim reaguje na ove događaje. Na primer, kada korisnik klikne jedan od tastera miša, izvršava se događaj. Poruka koja opisuje događaj šalje se prozoru koji se trenutno nalazi ispod pokazivača miša. Kod programa za taj prozor koji reaguje na događaje prihvatiće događaj, obraditi ga i odgovoriti na odgovarajući način. Kada program završi sa odgovaranjem na događaj, vraća se u stanje čekanja ili stanje bez posla (idle).

Kako ovo objašnjenje pokazuje, događaji su prešli na serijski prenos (serialized); svaki događaj se obrađuje tek pošto se prethodni događaj završi. Kada aplikacija izvršava kod za obradu događaja (to jest, kada ne čeka događaj), drugi događaji za tu aplikaciju moraju da čekaju u redu poruka rezervisanom za tu aplikaciju (izuzev ukoliko aplikacija koristi više procesa, kada svaki ima svoj red poruka). Kada je aplikacija odgovorila na poruku i kada se vratila u stanje čekanja, ona postaje poslednja aplikacija u listi programa koji čekaju da obrade dodatne poruke. Kod 16-bitnih Windowsa nije postojao način da se prekine aplikacija koja izvršava složenu obradu događaja, a ostale aplikacije su jednostavno morale da čekaju.

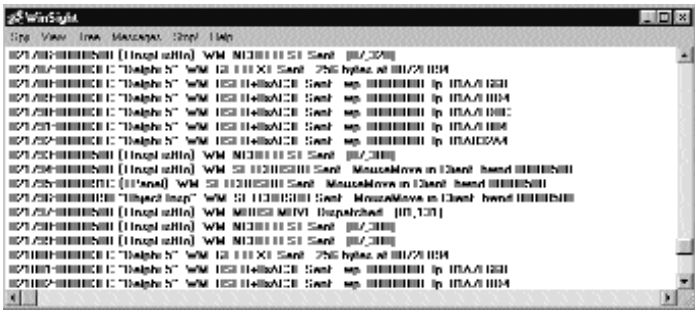
Obrađivanje događaja i redovi poruka su i dalje srž platforme Win32, ali u Windowsu 9x i Windowsu NT, pošto protekne određeno vreme, sistem prekida trenutnu aplikaciju i odmah predaje kontrolu narednoj aplikaciji koja se nalazi u listi. Prvi program nastavlja rad tek kada svaka od aplikacija dobije svoje vreme za izvršavanje. Ovo se naziva preemptive multitasking (prekidna višeprocesna obrada zadataka).

Zbog ograničene neprekidne višeprocesne obrade zadataka (nonpreemptive multitasking), Win16 aplikacije su koristile mnoge različite tehnike da bi pokušale da algoritam podele u manje delove i da ih tada izvršavaju jedan po jedan. Ove tehnike su uključivale upotrebu tajmera i obavljanje *idle* izvršavanja, a i dalje su korisne za ostvarivanje izvršavanja u pozadini a da nije potreban dodatni napor za upotrebu procesa. Zbog toga ću ih opisati u narednim odeljcima.

U Win32, ukoliko je aplikacija odgovorila na svoje događaje i čeka svoj red da bi obradila poruku, nema nikakve šanse da ponovo dobije kontrolu dok ne dobije narednu poruku (izuzev ukoliko se ne koristi više procesa). To je jedan od razloga zbog čega se tajmeri i dalje koriste. Na

kraju — kada razmišljate o događajima, imajte na umu da se ulazni događaji (upotrebom miša ili tastature) odnose samo na mali deo toka poruka u Windows aplikaciji. Većina poruka su interne sistemske poruke ili poruke koje se razmenjuju između različitih kontrola i prozora. Čak i poznata operacija unosa (kakva je klik tasterom miša) može da proizvede veliku količinu poruka, od kojih većinu čine interne Windows poruke.

Ovo i sami možete da testirate upotrebom pomoćnog programa WinSight koji je deo Delphija. U WinSightu odaberite da prikažete Message Trace, a zatim odaberite poruke za sve prozore. Odaberite Start, a zatim obavite uobičajene operacije mišem. Videćete stotine poruka u nekoliko sekundi (kao na slici 17.1). Naravno, WinSight prouzrokuje da se Windows izvršava sporije nego obično jer ga nadgleda. Pri normalnoj brzini tok poruka je mnogo brži od brzine koju ste videli kada izvršavate WinSight.



SLIKA 17.1 *Praćenje Windows poruka upotrebom alata WinSight koji je deo Delphija*

Prosleđivanje Windows poruka

Pre nego što pogledamo neke primere, potrebno je da razmotrimo još jedan važan element obrade poruke. Windows ima dva različita načina za slanje poruke prozoru:

- API funkcija `PostMessage` koristi se za smeštanje poruke u red poruka aplikacije. Poruka će biti obrađena tek kada aplikacija dobije priliku da pristupi svom redu poruka (to jest, kada joj sistem preda kontrolu) i tek kada su prethodne poruke obrađene. Ovo je asinhronizovani poziv jer ne znate kada će poruka zapravo biti primljena.
- API funkcija `SendMessage` koristi se za momentalno izvršavanje koda obrade poruke. `SendMessage` zaobilazi red poruka aplikacije i poruku šalje direktno u odredišni prozor ili kontrolu. Ovo je sinhronizovani poziv. Ova funkcija čak ima i povratnu vrednost, koju prosleđuje kod obrade poruke. Pozivanje funkcije `SendMessage` se ne razlikuje od direktnog poziva nekog metoda ili funkcije programa.

Izvršavanje u pozadini i multitasking

Pretpostavimo da morate da implementirate algoritam koji zahteva mnogo vremena za izvršavanje. Ukoliko algoritam napišete kao odgovor na događaj, Vaša aplikacija će potpuno biti zaustavljena sve vreme koliko je potrebno da se algoritam izvrši. Da bi se korisnik obavestio da

se nešto obrađuje, možete prikazati kursor u obliku peščanog sata, ali ovo rešenje nije blisko korisniku. Win32 omogućava ostalim programima da nastave izvršavanje, ali će se Vaš program blokirati; neće čak ažurirati ni svoj korisnički interfejs ukoliko se zatraži ponovno iscrtavanje. Zapravo, dok se algoritam izvršava, aplikacija neće moći da prihvati i obradi bilo koju drugu poruku, uključujući i poruku za iscrtavanje.

Najjednostavnije rešenje ovog problema je da pozovete metod `ProcessMessage` objekta Application mnogo puta u okviru algoritma, obično unutar interne petlje. Ovaj poziv zaustavlja izvršavanje, omogućava programu da primi i obradi poruku, a zatim da nastavi izvršavanje. Ova tehnika je bila korišćena u prethodnim poglavljima (kao u primerima Credits i Callback u Poglavlju 10), te je neću ovde prikazati. Problem kod ovakvog pristupa je u tome što — dok je program zaustavljen da bi prihvatio poruku — korisnik može da obavi bilo kakvu operaciju i može da klikne kontrolu ili pritisne tastere koji su pokrenuli algoritam. Da biste ovo popravili, možete da onemogućite tastere i komande koje ne želite da korisnik upotrebljava, i možete da prikažete kursor u obliku peščanog sata (koji, tehnički govoreći, ne sprečava klik mišem, ali sugeriše da korisnik treba da sačeka pre nego što uradi bilo koju drugu operaciju). Alternativno rešenje je da program podelite na manje delove i da te delove izvršavate jedan za drugim, omogućavajući aplikaciji da, između obrade delova, odgovori na poruke koje čekaju u redu poruka. U mnogim primerima prethodnih poglavlja (uključujući primer MdEdit5 u Poglavlju 7, primer LockTest u Poglavlju 10 i komponentu MdClock u Poglavlju 13) videli smo da možemo da upotrebimo tajmer da bismo omogućili sistemu da nas obavesti o vremenu koje je proteklo. Mada možete koristiti tajmere da biste implementirali neku vrstu izračunavanja u pozadini, ovo je daleko od dobrog rešenja. Nešto bolja tehnika je izvršavanje svakog koraka programa kada objekat `Application` primi događaj `OnIdle`.

Razlika između pozivanja funkcije `ProcessMessage` i upotrebe događaja `OnIdle` je u tome da ćete pozivanjem funkcije `ProcessMessage` svom kodu dati više vremena za procesiranje nego kada koristite metod `OnIdle`. Pozivanje funkcije `ProcessMessage` je način da sistem obavi ostale operacije dok Vaš program obavlja izračunavanja; upotreba događaja `OnIdle` je način da se aplikaciji omogući da obavi poslove u pozadini kada nema korisničkih zahteva koji čekaju.

Treći način za implementiranje izračunavanja u pozadini — manje uobičajen i manje složen — je taj da aplikacija pošalje korisničku poruku samoj sebi na kraju svakog koraka u izračunavanju u pozadini:

```
PostMessage (Handle, wm_User, 0, 0);
```

Aplikacija će ovu poruku dobiti posle nekog vremena, tako da može da izvrši naredni korak, a zatim će samoj sebi poslati narednu poruku, nastavljajući sve dok se izračunavanje u pozadini ne završi.

NAPOMENA

Sve ove tehnike za izračunavanje u pozadini bile su neophodne u danima 16-bitnih Windowsa. U platformi Win32, multitasking između aplikacija bolje funkcioniše, a sistem obezbeđuje procese baš da bi Vam omogućio da implementirate izračunavanje u pozadini. Ipak, tehnike kakva je upotreba tajmera, procesiranje događaja `OnIdle` i slanje korisničkih poruka su još uvek česti. ■

Provera da li postoji prethodna instanca aplikacije

Jedan oblik multitaskinga je izvršavanje dveju ili više instanci iste aplikacije. U opštem slučaju, korisnik može bilo koju aplikaciju da izvršava u više instanci i potrebno je da ima mogućnost da proveri postoji li još neka instanca koja se već izvršava, da bi aplikacija mogla da onemogući ovo unapred određeno ponašanje i da bi mogla da dozvoli samo jednu instancu. Ovaj odeljak prikazuje nekoliko načina implementiranja ovakve provere, što mi omogućava da razmatram brojne interesantne Windows programske tehnike.

NAPOMENA

U 16-bitnim Windows aplikacijama mogli ste da testirate vrednost sistemskog prametra `HPrevInstance` da biste videli da li se već izvršava instanca aplikacije. Na nesreću, u 32-bitnim Windows aplikacijama ovaj parametar je uvek 0. ■

Pronalaženje kopije glavnog prozora

Da biste pronašli kopiju glavnog prozora prethodne instance, upotrebite API funkciju `FindWindow` i prosledite joj naziv klase prozora (naziv koji ste koristili za registraciju tipa prozora formulara, ili `WNDCLASS`, u sistemu) i naslov prozora koji tražite. U Delphi aplikacijama naziv `WNDCLASS` klase prozora je isti kao i Object Pascal naziv za klasu formulara (na primer, TForm1). Rezultat funkcije `FindWindow` je ili hendl prozora ili nula (ukoliko se ne pronađe prozor).

Glavni kod Vaše aplikacije bi trebalo da bude napisan tako da se izvršava samo ukoliko je rezultat funkcije `FindWindow` nula:

```
var
  Hwnd: THandle;
begin
  Hwnd := FindWindow ('TForm1', nil);
  if Hwnd = 0 then
  begin
    Application.Initialize;
    Application.CreateForm (TForm1, Form1);
    Application.Run;
  end
  else
    SetForegroundWindow (Hwnd)
end.
```

Da biste aktivirali prozor prethodne instance aplikacije, možete da upotrebite funkciju `SetForegroundWindow`, koja funkcioniše za prozore koje poseduju neki drugi procesi. Ovaj poziv proizvodi efekat samo ukoliko je prozor koji je prosleđen kao parametar minimizovan. Kada je glavni formular Delphi aplikacije minimizovan, on je sakriven i zbog toga kod aktiviranja nema efekta.

Na nesreću, ukoliko pokrenete program koji koristi poziv `FindWindow` koji je upravo prikazan iz Delphi IDE-a, prozor sa datim naslovom i klasom može već da postoji: formular u vreme dizajniranja. Zbog toga program neće ponovo početi da se izvršva. Ipak, moći će da se pokrene ukoliko zatvorite formular i odgovarajući fajl sa izvornim kodom (zatvarajući samo formular jednostavno sakrivate prozor), ili ukoliko zatvorite projekat i pokrenete program iz Windows Explorera.

Upotreba muteksa (mutex)

Potpuno drugačiji pristup je upotreba *muteksa* ili objekata koji se uzajamno isključuju. Ovo je tipičan Win32 pristup, uobičajen za sinhronizovanje procesa, kao što ćemo videti kasnije u ovom poglavlju. Ovde ćemo upotrebiti muteks za sinhronizaciju dve različite aplikacije, ili (da budem precizniji) dve instance iste aplikacije.

Kada je aplikacija kreirala muteks sa zadatim nazivom, ona može testirati da li je taj objekat u vlasništvu neke druge aplikacije, pozivanjem Windows API funkcije `WaitForSingleObject`. Ukoliko muteks nema vlasnika, aplikacija koja poziva ovu funkciju postaje vlasnik. Ukoliko muteks već ima vlasnika, aplikacija čeka dok ne istekne time-out (drugi parametar aplikacije). Tada kao rezultat daje kod greške.

Da biste implementrali ovu tehniku, možete da upotrebite sledeći izvorni kod projekta, koji možete naći u primeru OneCopy:

```
var
  hMutex: THandle;
begin
  hMutex := CreateMutex (nil, False, 'OneCopyMutex');
  if WaitForSingleObject (hMutex, 0) <> wait_TimeOut then
  begin
    Application.Initialize;
    Application.CreateForm (TForm1, Form1);
    Application.Run;
  end;
end.
```

Ukoliko dva puta pokrenete primer, videćete da on kreira novu, privremenu kopiju aplikacije (pojavljuje se ikona na Taskbaru), a zatim se kopija uklanja kada istekne time-out. Ovaj pristup je svakako robusniji nego prethodni, ali mu nedostaju mogućnosti: kako da aktiviramo postojeću instancu aplikacije? Još uvek je potrebno da pronađemo formular, ali za to možemo da upotrebimo bolji pristup.

Pretraživanje liste prozora

Kada želite da pronađete određeni glavni prozor u sistemu, možete da upotrebite API funkciju `EnumWindows`. Funkcije enumeracije su prilično čudne u Windowsu jer obično zahtevaju drugu funkciju kao parametar. Ove funkcije enumeracije zahtevaju pokazivač na funkciju kao parametar (često se opisuje kao *callback* — povratna funkcija). Ideja je da se ova funkcija primenjuje za svaki element liste (u ovom slučaju, na listu prozora), sve do kraja liste ili dok funkcija ne vrati vrednost `False`. Evo funkcije enumeracije primera OneCopy:

```
function EndWndProc (hwnd: THandle;
  Param: Cardinal): Bool; stdcall;
var
ClassName, WinModuleName: string;
WinInstance:  THandle;
begin
  Result := True;
  SetLength (ClassName, 100);
  GetClassName (hwnd, PChar (ClassName), Length (ClassName));
```

```
  ClassName := PChar (ClassName);
  if ClassName = TForm1.ClassName then
  begin
    // get the module name of the target window
    SetLength (WinModuleName, 200);
    WinInstance := GetWindowLong (hwnd, GWL_HINSTANCE);
    GetModuleFileName (WinInstance,
      PChar (WinModuleName), Length (WinModuleName));
    WinModuleName := PChar(WinModuleName); // adjust length

    // compare module names
    if WinModuleName = ModuleName then
    begin
      FoundWnd := Hwnd;
      Result := False; // stop enumeration
    end;
  end;
end;
```

Ova funkcija, koja se poziva za svaki ne-dete prozor sistema, proverava naziv svake klase prozora, tražeći naziv klase `TForm1`. Kada pronađe prozor sa ovim stringom u nazivu klase, funkcija koristi `GetModuleFilename` za izdvajanje naziva izvršnog fajla aplikacije koja poseduje formular. Ukoliko se naziv modula podudara sa aktuelnim programom (koji je prethodno izdvojen sličnim kodom), možete biti prilično sigurni da ste pronašli prethodnu instancu istog programa. Evo kako možete da pozovete funkciju:

```
var
  FoundWnd: THandle;
  ModuleName:  string;

begin
  if WaitForSingleObject (hMutex, 0) <> wait_TimeOut then
    ...
  else
  begin
    // get the current module name
    SetLength (ModuleName, 200);
    GetModuleFileName (HInstance,
      PChar (ModuleName), Length (ModuleName));
    ModuleName := PChar (ModuleName); // adjust length

    // find window of previous instance
    EnumWindows (@EnumWndProc, 0);
```

Obrada korisničkih poruka prozora

Ranije sam pomenuo da poziv `SetForegroundWindow` ne funkcioniše ukoliko je glavni formular programa minimizovan. Sada možemo da rešimo taj problem. Možete da zatražite da formular iz druge aplikacije, u ovom slučaju prethodne instance istog programa, restaurira glavni formular slanjem korisničke poruke prozora. Zatim možete testirati da li je formular minimizovan i možete poslati novu korisničku poruku starom prozoru. Ovde se nalazi kod; u programu OneCopy ovaj kod se nalazi iza poslednjeg fragmenta koji je prikazan u prethodnom odeljku:

```
if FoundWnd <> 0 then
begin
  // show the window, evenrually
  if not IsWindowVisible (FoundWnd) then
    PostMessage (FoundWnd, wm_User, 0, 0);
  SetForegroundWindow (FoundWnd);
end;
```

Ponoviću, API funkcija `PostMessage` šalje poruku u red poruka aplikacije koja poseduje odredišni prozor. U kodu formulara možete dodati specijalnu funkciju koja obrađuje ovu poruku:

```
public
  procedure WMUser (var msg: TMessage):
    message wm_User;
```

Sada možete napisati kod ovog metoda, koji je, zapravo, jednostavan:

```
procedure TForm1.WMUser (var msg: TMessage);
begin
  Application.Restore;
end;
```

Više procesa u Delphiju

Win32 nam omogućava da se dve procedure ili dva metoda istovremeno izvršavaju, i da omogućimo programu da ih kontroliše. Pre nego što pogledamo implementaciju više procesa, potrebno je da se zapitamo zbog čega bi nam bilo potrebno više procesa izvršavanja unutar datog programa. Najpre razmotrimo neke nedostatke više procesa:

- Pokretanje više procesa čini da se program izvršava sporije, izuzev ukoliko nemate više CPU-ova i ukoliko operativni sistem može da razdeli procese među procesorima.
- Loše napisana aplikacija koja koristi više procesa može sporije da se izvršava na višeprocesorskim nego na jednoprocesorskim sistemima. Sinhronizacija između procesa je mnogo skuplja na višeprocesorskim sistemima.
- Programi sa više procesa moraju da sinhronizuju pristup deljenim resursima i memoriji, što njihovo pisanje čini komplikovanijim, kao što ćemo videti u mnogim klasama.

Na sreću, više procesa ima neke prednosti. Na primer, proces možete da izvršavate u pozadini, omogućavajući korisniku da nastavi da operiše programom. Možete učiniti da se jedan proces izvršava brže od drugih ukoliko podesite njegov prioritet, regulišete pristup resursima drugih procesa, dodelite lokalno čuvanje za svaki proces i generišete više procesa istog tipa. Međutim, ključna prednost je u tome da možete jednostavno da napišete algoritam unutar procesa, a da ne morate brinuti kako da ga podelite, omogućavajući sistemu da osveži korisnički interfejs, ili da učini bilo šta drugo.

Najbolje je u pozadinu smestiti proces koji zahteva prilično procesorskog vremena za izvršavanje (za operativni sistem je veoma skupo da formira procese, tako da morate da budete sigurni da se to isplati) i prilično je izolovan kada se uzme u obzir pristup podacima (nema sinhronizacije za pristup deljenim resursima). Ukoliko se operacija brzo obavlja, ili umnogome

zavisi od spoljašnjih podataka, ne treba se mučiti oko procesa. Jednostavno, upotreba procesa za deljenje procesa od više koraka (koji je po prirodi sekvencijalan) je gubljenje vremena. (Moji primeri programa se ne pridržavaju uvek ovih pravila; ali, to su samo primeri načinjeni da Vam demonstriraju određene tehnike.)

Klasa TThread

Windows obezbeđuje niz API poziva za kontrolu procesa (ključni poziv je `CreateThread`), ali ih ovde neću razmatrati jer Delphi obezbeđuje klasu `TThread` koja će nam omogućiti da veoma dobro kontrolišemo proecese.

Prva stvar koju treba da znate o klasi `TThread` jeste da je nikada direktno ne koristite, jer je to apstraktna klasa — klasa sa virtuelnim apstraktnim metodom. Da biste upotrebili procese, uvek pravite potklase klase `TThread` i koristite karakteristike osnovne klase. Klasa `TThread` sadrži konstruktor sa jednim parametrom koji Vam omogućava da odlučite da li da proces odmah pokrenete, ili da proces sačeka sa izvršavanjem:

```
constructor Create (createSuspended: Boolean);
```

Postoji takođe i nekoliko metoda sinhronizacije:

```
procedure Resume;
procedure Suspend;
function Terminate: Integer;
function WaitFor: Integer;
```

Published svojstva uključuju `Priority`, `Suspend` i dve vrednosti niskog nivoa samo za čitanje: `Handle` i `ThreadID`. Klasa takođe obezbeđuje zaštićeni interfejs, koji sadrži dva ključna metoda za procese potklasa:

```
procedure Execute; virtual; abstract;
procedure Synhronize (Method: TThreadMethod);
```

Metod `Execute`, koji je deklarisan kao virtuelna apstraktna procedura, mora biti ponovo definisan za svaku klasu procesa. On sadrži glavni kod procesa, kod koji biste obično smestili u funkciju procesa (thread function) kada koristite Windows API. Metod `Synhronize` se koristi da bi se izbegao konkurentan pristup VCL komponentama. VCL kod se izvršava unutar glavnog procesa programa, i potrebno je da sinhronizujete pristup VCL-u da biste izbegli probleme ponovnog ulaska (greške kada se u funkciju ponovo ulazi, a da prethodni poziv nije završen). Jedini parametar metoda `Synhronize` je metod koji ne prihvata parametre, obično metod iste klase procesa.

Opšte VCL kontrole, nasuprot tome, nisu bezbedne od procesa (thread-safe). Na primer, ne bi trebalo da kreirate VCL kontrole u kontekstu procesa u pozadini: prozor obrađuje samo poruke iz reda poruka procesa u kome je hendl kreiran. Ukoliko kreirate kontrolu u procesu u pozadini, objekat `Application` u glavnom procesu neće proslediti poruke hendlu prozora kontrole. Microsoft je izričito protiv upotrebe više procesa za kreiranje korisničkog interfejsa.

Ono što VCL podržava, ograničeno doduše, jeste manipulacija glavnim procesom VCL kontrola iz procesa u pozadini. Najčešći slučaj je proces u pozadini koji želi da iscrta formular. Na primer, klasa `TCanvas` sadrži dva metoda zaključavanja, `Lock` i `UnLock`, koji omogućavaju procesu u pozadini da direktno iscrtava na `Canvas` glavnog formulara. Klase `TBitmap` i `TJPEGImage` se mogu

koristiti za obradu slika u procesima u pozadini. Još jedna korisna klasa je `TThreadList`, koja omogućava različitim procesima da pristupe istoj `TList` klasi bezbedno i konkurentno.

SAVET

Delphi 5 je poboljšao sigurnost procesa delova izvršne biblioteke (run-time library — RTL). Prebrojavanje referenci stringova sada funkcioniše između procesa, a prebrojavanje referenci COM objekata je takođe poboljšano, radi bolje podrške COM apartment-proces modela. ■

Prvi primer

Za prvi primer sam izradio program koji koristi metod `Synhronize`. Program, nazvan ThOld, koristi proces za iscrtavanje po površini formulara. Klasa procesa, `TPainterThread`, zaobilazi metod `Execute` i definiše korisnički metod `Paint`. Metod `Paint` se koristi za pristupanje VCL objektima, tako da se poziva samo iz metoda `Syhnronization`. Pošto metod `Paint` ne može direktno da primi parametre i ostane kompatibilan argument metoda `Syhnronization`, klasi su potrebni neki privatni podaci. Evo deklaracije klase procesa:

```
type
  TPainterThread = class (TThread)
  private
    X, Y: Integer;
protected
    procedure Execute; override;
    procedure Paint;
end;
```

Metod `Paint` obeležava piksele formulara crvenom bojom:

```
procedure TPainterThread.Paint;
begin
  Form1.Canvas.Pixels [X, Y] := clRed;
end;
```

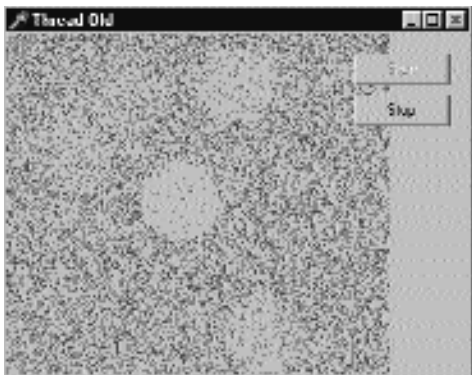
Glavna funkcija procesa, `Execute`, nasumično ažurira piksel prosleđivanjem metoda `Paint` kao parametra metoda `Synhronize`:

```
procedure TPainterThread.Execute;
begin
  Randomize;
  repeat
    X := Random (300);
    Y := Random (Form1.ClientHeight);
    Synchronize (Paint);
  until Terminated;
end;
```

Na slici 17.2 je prikazan rezultat ovog koda. Kao što iz prethodnog listinga možete videti, proces se izvršava sve dok se ne prekine. Na glavnom formularu se nalazi kontrola kojom se pokreće proces. Kako prosleđujemo vrednost `False` konstruktoru `Create` (metod `Button1Click`), proces momentalno započinje izvršavanje:

```
PT := TPainterThread.Create (False); // start immediatly
```

PT je privatno polje TPainterThread klase formulara. Druga kontrola oslobađa objekat procesa.



SLIKA 17.2 *Izlaz primera ThOld. Obojeni pikseli su iscrtani procesom koji se izvršava u pozadini*

Glavni formular takođe obrađuje događaje klika mišem; kada je pritisnut taster miša nad formularom, crveni pikseli unutar kruga oko mesta na kome ste kliknuli se brišu, bojeći formular bojom pozadine (što možete pretpostaviti ukoliko pogledate sliku 17.2):

```
procedure TForm1.FormMouseDown (Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  Canvas.Pen.Color := Color; // of the form
  Canvas.Brush.Color := Color;
  Canvas.Ellipse (x - 30, y - 30, x + 30, y + 30);
end;
```

Primer zaključavanja

Sada možemo iznova da napišemo prethodni primer koristeći osnovnu podršku za sinhronizaciju procesa koju obezbeđuje klasa TCanvas. U ovom slučaju, da bismo pristupili slici formulara, možemo jednostavno da je zaključamo i izbegnemo poziv metoda Synhronization, pojednostavljujući kod i čineći izvršavanje koda mnogo bržim. Klasa procesa u primeru ThLock ima samo jedan metod:

```
type
  TPainterThread = class(TThread)
  protected
    procedure Execute; override;
  end;
```

Kod metoda Execute sada radi sve, uključujući prikazivanje izlaza, posle zaključavanja slike formulara:

```
procedure TPainterThread.Execute;
var
  X, Y: Integer;
begin
  Randomize;
```

```
  repeat
    X := Random (300);
    Y := Random (Forml.ClientHeight);
    with Form1.Canvas do
    begin
      Lock;
      try
        Pixels [X, Y] := clRed;
      finally
        Unlock;
      end;
    end;
  until Terminated;
end;
```

Ukoliko zatim želite da pristupite `Canvasu` iz neke druge obrade događaja sem `OnPaint` (koja automatski rukuje zaključavanjem), trebalo bi da ponovo pozovete metod `Lock`:

```
procedure TForm1.FormMouseDown (Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  Canvas. Lock;
  try
    Canvas.Pen.Color := clYellow;
    Canvas.Brush.calar  clYellow;
    Canvas.Ellipse (a - 30, y - 30, a 30, y + 30);
  finally
    Canvas Unlock;
  end;
end;
```

Alternative sinhronizacije

Upotreba metoda Lock je ograničena na nekoliko Delphi objekata koji imaju ovu mogućnost. Alternativa je da nastavite da koristite zaštićeni metod `Synhronize` klase `TThread` kada očekujete da nekoliko procesa istovremeno ima potrebu da pristupi svojstvima komponente. U opštem slučaju biste koristili sekundarne procese za operacije u pozadini, kao što su transfer fajlova ili operacije sa brojevima, a da gotovo i nema potrebe za ažuriranjem korisničkog interfejsa. Ukoliko su Vam potrebna ograničena ažuriranja korisničkog interfejsa, postoje alternativni pristupi dvema tehnikama koje sam pomenuo.

Prvo, proces može da ažurira neke strukture podataka (kao što su red ili kružni bafer), koje glavni proces pretražuje s vremena na vreme. Morate se postarati da izbegnete kolizije pisanja/čitanja između različitih procesa. Druga alternativa je upotreba tradicionalnog Windows multitaskinga; proces može da pošalje poruku glavnom prozoru tražeći ažuriranje. Imajte na umu da nećete uvek moći da upotrebite `SendMessage` (koji je sinhronizovan i sličan direktnom pozivu funkcije); umesto toga, trebalo bi da koristite samo `PostMessage` (koji je asinhronizovan i koristi red poruka). Ove alternativne tehnike se ne koriste često u poređenju sa prve dve tehnike koje sam Vam pokazao, pa zato ne postoje primeri u knjizi koji implementiraju ove tehnike.

Prioritet procesa

Naš treći primer procesa je proširenje prethodnog primera. Ovoga puta koristimo nekoliko procesa istovremeno, a program omogućava korisnicima da promene prioritete procesa upotrebom nekoliko linija. Evo nove verzije klase TPainterThread:

```
type
  TPainterThread = class(TThread)
  private
    Color:  Integer;
  protected
    procedure Execute; override;
  public
    constructor Create (Col: TColor);
  end;
```

Klasi sam dodao konstruktor da bih prosledio početnu vrednost boje procesu. Kao alternativu, mogao sam da načinim polje `Color` javnim poljem klase procesa da bih programu omogućio da direktno manipuliše poljem. Evo koda konstruktora:

```
constructor TPainterThread.Create(Col: TColor);
begin
  Color:= Col;
  inherited Create (True);
end;
```

Konstruktor inicijalizuje privatne podatke, a zatim poziva konstruktor osnovne klase, kreirajući proces u suspendovanom stanju. Metod `Execute` procesa jednostavno skenira svaku liniju ekrana, dodeljujući svakom pikselu navedenu boju:

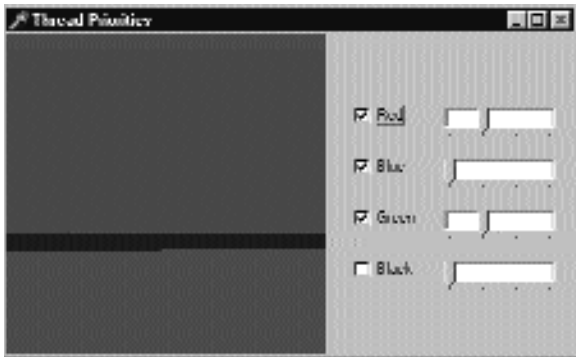
```
procedure TPainterThread. Execute;
var
  X, Y, X1: Integer;
begin
  X := 0;
  Y := 0;
  repeat
    // scan the lines...
    X1 := X + 1;
    X := X1 mod 250;
    Y := Y + X1 div 250;
    Y := Y mod Form1.ClientHeight;
    Form1.Canvas . Lock;
    try
      Form1.Canvas.Pixels [X, Y] := Color;
    finally
      Form1.Canvas.UnLock;
    end;
  until Terminated;
end;
```

Glavni formular sadrži četiri polja za potvrdu i četiri linije (track bar), kao što možete videti na slici 17.3. Formular takođe sadrži i neke lokalne podatke, niz za čuvanje četiri objekta procesa:

```
private
  PT: array [1. .4] of TPainterThread;
```

Ovaj niz se inicijalizuje prilikom kreiranja formulara:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  PT [1] := TPainterThread.Create (clRed);
  PT [2] := TPainterThread.Create (clBlue);
  PT [3] := TPainterThread.Create (clGreen);
  PT [4] := TPainterThread.Create (clBlack);
end;
```



SLIKA 17.3 *Izlaz primera ThPrior, kada četiri procesa konkurentno ažuriraju korisnički interfejs*

Primetićete da program kreira četiri procesa kao suspendovane procese. Procesi započinju izvršavanje kada se potvrdi odgovarajuće polje i ponovo se suspenduju kada polje nije potvrđeno:

```
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  if (Sender as TCheckbox).Checked then
    PT [(Sender as TCheckbox).Tag].Resume
  else
    PT [(Sender as TCheckbox)Tag].Suspend;
end;
```

Da bih upotrebio istu obradu događaja za sva polja za potvrdu, ja sam odredio vrednost svojstva `Tag` za svako polje tako da odgovara broju odgovarajućeg procesa. Ponekad, kada isključite neki od procesa selektovanjem odgovarajućeg polja za potvrdu, svi procesi će se istovremeno zaustaviti. Ukoliko se desi da zaustavite proces dok je zaključao sliku, svi ostali procesi će morati da čekaju da se slika otključa, a to se ne može desiti sve dok se ne nastavi proces. Ovaj problem može da se reši na razne načine, upotrebom Canvasovih metoda `TryLock` i `LockCount`. U konačnoj verziji primera ThPrior ja sam zamenio poziv metoda `Suspend` pozivom jedne druge funkcije koju sam dodao procesu:

```
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  if (Sender as TCheckbox).Checked then
    PT [(Sender as TCheckbox).Tag].Resume
  else
    PT [(Sender as TCheckbox).Tag].DelayedSuspend;
end;
```

Metod `DelayedSuspend` procesa jednostavno podešava Boolean vrednost `SuspendRequest` polja procesa. Ova Boolean vrednost se proverava na kraju cilkusa unutar izvršnog koda procesa, koji u pravo vreme poziva metod `Suspend`. Sledi kod koji je dodat metodu `TPainterThread.Execute` (dodat je listingu koji je ranije prikazan, upravo pre kraja petlje `repeat-until`):

```
if SuspendRequest then
begin
  Suspend;
  SuspendRequest := False;
end;
```

Ovom tehnikom sigurno znamo da je metod `Suspend` pozvan samo kada proces ne zaključava sliku.

Svojstvo `Tag` se takođe koristi uz linije, kojim se određuje trenutni prioritet procesa:

```
procedure TForm1.TrackBar1Change (Sender: TObject);
begin
  PT [(Sender as TTrackBar).Tag]. Priority :=
    TThreadPriority ((Sender as TTrackBar).Position);
end;
```

Da bih odredio prioritet, ja sam jednostavno konvertovao `Position` linije u odgovarajuću numerisanu vrednost `TThreadPriority`. Zatim sam rezultujuću vrednost upotrebio za određivanje prioriteta odgovarajućeg procesa, kao što je određeno svojstvom `Tag`. Ovaj primer je veoma poučan jer možete da promenite prioritet procesa, a efekat možete da vidite na ekranu.

Dok izvršavate primer, primetićete da će proces koji ne blokira i koji se izvršava sa najvišim prioritetom (makar samo jedan stepenik iznad normalnog), zauzeti gotovo sve vreme CPU-a i usporiti sve procese nižeg prioriteta. To znači da treba samo da povećate prioritet procesa koji veći deo vremena provede blokiran, čekajući signal; čak i tada, trebalo bi da razmislite da li da se poigrate prioritetom procesa. U Windows aplikacijama je uobičajenije da smanjite prioritet procesa ispod normale, tako da se izvršava tek kada se sve drugo obavi i kada aplikacija počne da odgovara na ulaz korisnika.

Sinhronizovanje procesa

Videli smo da postoje dva uobičajena pristupa sinhronizaciji procesa sa ostatkom aplikacije: upotrebom metoda `Synhronization` objekta procesa i upotrebom metoda `Lock` VCL klase koja ga obezbeđuje, kao što je klasa `TCanvas`. U jednom od ranijih primera smo takođe koristili metod `Lock` za `Canvas` glavnog formulara da bismo sinhronizovali četiri procesa koja su iscrtavala ekran. Međutim, to nije bio tipičan slučaj. U opštem slučaju biće potrebno da koristite tehnike za sinhronizaciju dva procesa, uključujući tehnike niskog nivoa koje su na raspolaganju u Windows API-ju.

U narednom odeljku ću razmatrati jednostavan slučaj, čekajući da se proces prekine. Zatim ćemo, u narednim odeljcima, videti komplikovanije primere.

Čekanje na proces

Kada proces treba da čeka da se obavi neki drugi proces, može jednostavno pozvati metod `WaitFor` objekta koji odgovara procesu koji treba da se prekine. Evo dela primera, u kojem program započinje proces, a zatim čeka rezultate:

```
Comp :=  TMyThread.Create (True);
// initialize the thread ...
Comp.Resume;
Comp.Wait;
// look for final values ...
Comp.Free;
```

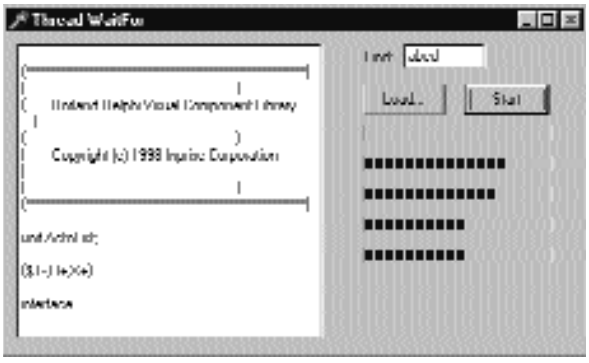
UPOZORENJE

Ukoliko kreirate proces u suspendovanom stanju, a zatim ga uklonite pre nego što nastavite njegovo izvršavanje, možete iskusiti nedostatak memorije. Ovo nije samo problem u Delphiju; Microsoftovi alati Vas takođe upozoravaju. Tehnički razlog je da kreiranje hendla procesa uključuje privremeno alociranje memorije za prosleđivanje kontekst informacija u proceduru koja je pridružena procesu. Čim proces započne izvršavanje, privremena memorija se oslobađa; ali, kada se proces ukloni a da nikada ne počne da se izvršava, privremena memorija je izgubljena. Količina izgubljene memorije je prilično mala (samo 12 bajtova), ali čak i ovo se može pretvoriti u katastrofalnu situaciju u server aplikaciji koja treba da se neprekidno izvršava mesecima ili godinama. ■

Ovaj kod je prilično jednostavno napisati, ali zapamtite da ovaj kod ne možete napisati kao deo glavnog procesa (na primer, u normalnoj funkciji koja odgovara na poruke) ukoliko sekundarni proces treba da se sinhronizuje. Ako proces povezan sa glavnim formularom čeka završetak nekog drugog procesa, a sekundarni proces čeka pristup korisničkom interfejsu (te zbog toga čeka da glavni proces završi posao koji trenutno obavlja), program će se blokirati!

Da biste izbegli ovaj problem, možete da upotrebite metod `WaitFor` za sinhronizaciju dva procesa. Prvi proces kreira sekundarni proces, a zatim čeka da se proces završi bez smetanja glavnom procesu.

Da bih Vam pokazao primer sinhronizacije sa više procesa, izradio sam program koji prebrojava karaktere i nazvan je ThWait. Program izračunava koliko kopija četiri karaktera naznačenih u polju za izmene postoji u tekstu Memo komponente (kao alternativu, možete da učitate tekst iz bilo kog fajla, ukoliko izvršite izračunavanje pre pokretanja). Program istovremeno traži bilo koji od četiri karaktera, koristeći više procesa generisanih glavnim procesom. Da bi poboljšao izlaz, svaki proces u progres liniji prikazuje svoj status — to jest, koliko teksta je pretražio — kao što možete videti na slici 17.4.



SLIKA 17.4 *U primeru ThWait svaki proces prikazuje svoj status na progres liniji*

Kako se do memo linija dolazi upotrebom poruke SendMessage za hendl prozora kontrole, a poruka prozora se procesira u kontekstu glavnog procesa, u praksi ne postoji prednost upotrebe četiri procesa za skeniranje teksta iste memo kontrole. Ipak, primer demonstrira neke korisne tehnike.

Pravi mehanizam programa je klasa TFindThread, koja sadrži polje LookFor u kojem se čuva karakter koji tražimo i polje Progress u kojem se čuva vrednost progres linije i ažurira status. Rezultat izračunavanja se smešta u javno polje Found:

```
type
  TFindThread = class (TThread);
  protected
    Progr: Integer;
    procedure UpdateProgress;
    procedure Execute; override;
  public
    Found: Integer;
    LookFor: Char;
    Progress: TProgressBar;
end;
```

Kao i obično, srž procesa se nalazi u njegovom metodu Execute, koji pretražuje linije beleške u potrazi za zadatim karakterom. Primetićete da slobodno možemo pristupati svojstvima beleške, a da ne moramo da vršimo sinhronizaciju, jer ova operacija nije destruktivna — ne utiče na status Memo komponente:

```
procedure TFindThread.Execute;
var
  I, J: Integer;
  Line: string;
begin
  Found := 0;
  with Form1.Memo1 do
  for I := 0 to Lines.Count - 1 do
    begin
      Line := Lines [I];
      for J := 1 to Length (Line) do
        if Line [J] = LookFor then
          inc (Found);
      Progr := I + 1;
      Synchronize (UpdateProgress);
    end;
end;
```

Metod UpdateProgress jednostavno ažurira status progres linije koristeći vrednost polja Progr:

```
procedure TFindThread.UpdateProgress;
begin
  Progress.Position := Progr;
end;
```

Četiri kopije ovog procesa se aktiviraju primarnim procesom, objektom klase TMultiFind. Evo deklaracije te klase:

```
type
  TMultiFind = class (TThread)
  protected
    Progr: Integer;
    procedure UpdateProgress;
    procedure Execute; override;
    procedure Show;
public
    LookFor, Output: String;
    Progress: array [1..5] of TProgressBar;
end;
```

Ova klasa procesa traži karaktere stringa `LookFor` (moraju da postoje četiri karaktera da bi program pravilno funkcionisao), koristeći četiri objekta `TFindThread`:

```
procedure TMultiFind. Execute;
var
  Finders: array [1. .4] of TFindThread;
  I: Integer;
  begin
  // set up the four threads
  for I := 1 to 4 do
  begin
    Finders[I] := TFindThread.Create (True);
    Finders[I].LookFor := LookFor[I];
    Finders[I].Progress := Progresses [I+1];
    Finders[I] Resume;
  end;
  // wait for the threads to end...
  for I := 1 to 4 do
  begin
    Finders[I].WaitFor;
    Progr := I;
    Synchronize (UpdateProgress);
  end;
  // show the result
  Output := 'Found: ';
  for I := 1 to 4 do
    Output := Output + Format ('%d %s, ' ,
      [Finders[I] Found, LookFor[I]]);
  Synchronize (Show);
  // delete threads
  for I := 1 to 4 do
    Finders[I] Free;
end;
```

Naročito obratite pažnju na petlju for u kojoj je poziv `WaitFor`. Na kraju, metod `Execute` prikazuje rezultat sinhronizovanog metoda — metoda `Show`. Program koji sam napisao da bih testirao ova četiri procesa je prilično jednostavan. Kao što ste videli na slici 17.4, program sadrži memo kontrolu u koju možete učitati fajl i polje za izmene koje sadrži četiri karaktera. Broj karaktera se proverava kada korisnik izađe iz polja za izmene:

```
procedure TForm1.Edit1Exit(Sender: TObject);
begin
  if Length (Edit1.Text) <> 4 then
  begin
    Edit1.SetFocus;
    ShowMessage ('The edit box requires four characters');
  end;
end;
```

Kontrola Start započinje proces, koji odmah generiše sekundarne procese:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  if Assigned (MainThread) then
    MainThread. Free;
  MainThread := TMultiFind.Create (True);
  MainThread.Progresses [1] := ProgressBarl;
  MainThread.Progresses [2] := ProgressBar2;
  ...
  MainThread.Progresses [1].Max := 4;
  for I := 2 to 5 do
    MainThread.Progresses[I].Max := Memo1.Lines.Count;
  for I := 1 to 5 do
    MainThread.Progresses[I].Position := 0;
  MainThread.LookFor := Edit1.Text;
  MainThread.Resume;
end;
```

Primetićete da ne možemo da uklonimo proces na kraju metoda, jer ne možemo pozvati metod **WaitFor** a da ne dovedemo do blokiranja programa. Dobro pripazite kada pišete aplikacije sa više procesa, jer ovakvo blokiranje može da blokira ceo sistem, ne ostavljajući Vam nijednu opciju sem da pritisnete kombinaciju tastera Ctrl+Alt+Del u Windowsu 95/98.

Istovremeno, da bismo sistem održali stabilnim, ne smemo da zaboravimo da uklonimo proces, bilo pre kreiranja drugog procesa (kao na početku prethodnog koda), bilo kada program prestane da se izvršava. Ovo je, takođe, razlog zbog koga moramo da deklarišemo objekat procesa kao privatno polje formulara, a ne kao lokalnu promenljivu metoda koji pokreće proces.

Windows tehnike sinhronizacije

Win32 API funkcije nude još mnogo tehnika sinhronizacije:

- *Kritični odeljci* (critical sections) su delovi izvornog koda koji se ne mogu izvršavati istovremeno u dva procesa. Upotrebom kritičnog odeljka možete preći na serijski prenos izvršavanja određenih delova izvornog koda. Kritični odeljci se mogu koristiti samo u okviru jednog procesa, jedne aplikacije.
- *Muteksi* (mutex) su globalni objekti koje možete da koristite za serijalizaciju pristupa resursima. Prvo podesite muteks, zatim pristupite resursu i na kraju oslobodite muteks, kao što smo videli u primeru OneCopy. Dok je muteks određen, ukoliko neki drugi proces (ili procesi) pokušaju da odrede isti muteks,

određivanje se zaustavlja sve dok muteks ne oslobodi prethodni proces. Muteks može biti deljen između različitih aplikacija.

- *Semafori* (semaphores) su veoma slični muteksima, ali se semafori prebrojavaju: na primer, možete omogućiti samo tri i ne više od tri istovremena pristupa datom resursu. Muteks je ekvivalent semaforu koji omogućava samo jedan pristup. Primer upotrebe semafora ćemo videti u odeljku "Procesni pristup bazi podataka", kasnije u ovom poglavlju.
- *Događaji* (events) se mogu koristiti kao način sinhronizacije procesa upotrebom sistemskih događaja, recimo u slučaju korisničkih operacija sa fajlovima. Metod `WaitFor` Delphi klase `TThread` koristi događaj. Događaji se mogu koristiti za pobuđivanje (awaken) nekoliko procesa u isto vreme.

Izrada primera

Da bih demonstrirao sve ove različite tehnike, izradio sam primer ThSynch. Pretpostavimo da imamo dva procesa koja operišu stringom, oba procesa koriste na neki način vrednost stringa, a zatim ažuriraju string rezultatom svoje operacije. Pretpostavimo, takođe, da je aktuelna vrednost stringa deljena između dva procesa. U primeru string inicijalno sadrži 20 A karaktera, zatim se ažurira da sadrži 20 B karaktera i tako dalje. Svaki proces izračunava narednu vrednost stringa i šalje ga svojoj listi.

NAPOMENA

U pravim aplikacijama, naravno, trebalo bi da procesi izbegavaju upotrebu globalnih promenljivih. Delphi pomaže u ovakvom slučaju omotavajući procese u klase i omogućavajući Vam da definišete promenljive procesa, upotrebom ključne reči `threadvar`. Program ThSynch je naročito izrađen da bi Vam prikazao probleme sinhronizacije; ne treba ga smatrati dobrim primerom deljenja podataka između procesa! ■

Primer ThSynch sadrži glavni formular sa četiri kontrole, od kojih svaka prikazuje sekundarni formular koji demonstrira jednu od tehnika sinhronizacije. Svaki sekundarni formular se sastoji od dve liste, u kojima se prikazuje tekst u fontu Courier, i kontrole za pokretanje odgovarajućeg procesa. Dakle, na kraju ćemo dobiti četiri različite verzije u osnovi istog formulara i istu klasu procesa. U svakom od formulara kontrola Start jednostavno kreira dve instance procesa, pridružujući listu javnom polju `LBox`:

```
procedure TForm2.BtnStartClick (Sender: TObject);
begin
  ListBox1.Clear;
  ListBox2.Clear;
  Th1 := TListThread.Create (True);
  Th2 := TListThread.Create (True);
  Th1.FreeOnTerminate := True;
  Th2.FreeOnTerminate := True;
  Th1.LBox := Listbox1;
  Th2.LBox := Listbox1;
  Th1.Resume;
  Th2.Resume;
end;
```

Primetićete da je određena vrednost `True` za svojstvo `FreeOnTerminate`, tako da se objekat procesa automatski oslobađa kada se završi obrada. Klase procesa su deklarisane u svakoj jedinici koja definiše formular, da bi se izbeglo postojanje prevelikog broja fajlova. Ova ista jedinica sadrži promenljivu koju koriste dva objekta procesa:

```
var
  Letters: string = 'AAAAAAAAAAAAAAAAAAAA';
```

Ovo je daleko od elegantnog rešenja, ali u ovom programu sa razlogom tražimo problem, te zaboravite na stil kodiranja.

Običan proces

Sledi prva jednostavna verzija klase procesa i njen metod `Execute`:

```
type
  TlistThread = class (TThread)
  private
    Str: String;
  protected
    procedure AddToList;
    procedure Execute; override;
  public
    LBox: TListBox;
  end;

procedure TListThread.Execute;
var
  I, J, K: Integer;
begin
  for I := 0 to 50 do
  begin
  for J:= 1 to 20 do
    for K:= 1 to 2601 do // useless repetition ...
      if Letters [J] <> 'Z' then
        Letters [J] :=Succ (Letters [J])
      else
        Letters [J] := 'A';
    Str := Letters;
    Synchronize (AddToList);
  end;
end;
```

Metod `AddToList` jednostavno dodaje string `Str` listi koja je pridružena procesu. Ja sam svako izračunavanje namerno učinio dugim, povećavajući svako slovo `2601` put umesto jednom: efekat je isti, ali postoje veće šanse za konflikt između dva procesa. Efekat ovog koda možete videti na slici 17.5. Još bolje, možete kliknuti kontrolu Start dva ili više puta zaredom, pokrećući odjednom nekoliko procesa, i povećavajući šanse da se dogodi greška.

SLIKA 17.5 *Prvi sekundarni formular primera ThSynch prikazuje neke greške u vrednostima dveju lista*

Upotreba kritičnih odeljaka

Ukoliko želite da serijalizujete operacije dva procesa i da imate veću kontrolu nad onim što procesi čine, možete upotrebiti jednu od Windows tehnika sinhronizacije, koje sam ranije razmatrao. U ovoj drugoj verziji ću upotrebiti kritične odeljke. Da biste to učinili, potrebno je da dodate deklaraciju još jedne globalne promenljive (ili polje klase formulara) za kritični odeljak:

```
var
  Critical1: TRTLCriticalSection;
```

Ova promenljiva se inicijalizuje prilikom kreiranja formulara, i uklanja se na kraju, upotrebom dva API poziva:

```
procedure TForm3.FormCreate (Sender: TObject(;
begin
  InitializeCriticalSection (Critical1);
end;

procedure TForm3.FormDestroy (Sender: TObject);
begin
  DeleteCriticalSelection (Critical1);
end;
```

SAVET

Kada koristite kritične odeljke, uvek se postarajte da se procesi koji zavise od kritičnih odeljaka završe pre nego što uklonite taj objekat sinhronizacije. ■

Možete koristiti kritične odeljke za serijalizaciju određenih delova koda, recimo, koda koji ažurira svako slovo stringa. Evo kako sam ja ažurirao kod metoda `Execute` objekta procesa:

```
procedure TListThread.Execute;
var
  I, J, K: Integer;
begin
  for I := 0 to 50 do
  begin
```

```
    EnterCriticalSelection (Critical1);
    try
      for J := 1 to 20 do
        ... // same code as above
      Str := Letters;
    finally
      LeaveCriticalSelection (Critical1);
    end;
    Synchronize (AddToList);
  end;
end
```

Efekat koji proizvodi ovaj kod je takav da dok jedan proces izračunava novi string, drugi proces čeka da bi obavio istu operaciju, tako da će svi izlazni stringovi uvek sadržati 20 kopija istog karaktera.

Upotreba muteksa

Sada možemo da napišemo isti kod, ali da upotrebimo muteks umesto kritičnog odeljka. Efekat će biti isti, ali bih želeo da Vam prikažem i ovu tehniku. Potrebno je da jednostavno deklarišemo promenljivu `hMutex` tipa `THandel` i da zatim zamenimo četiri API poziva prethodnog primera sledećim pozivima:

```
// in FormCreate
hMutex := CreateMutex (nil, false, nil);
// in FormDestroy
CloseHandle (hMutex);
// in the for loop
WaitForSingleObject (hMutex, INFINITE);
...
ReleaseMutex (hMutex);
```

Upotreba klase TCriticalSection VCL objekta

Enterprise izdanje Delphija nudi još jednu mogućnost. Jedinica SyncObjs definiše VCL klase za neke objekte sinhronizacije: događaje i kritične odeljke. Dakle, možemo da izradimo četvrtu verziju primera, veoma sličnu drugoj verziji, koja je bazirana na kritičnim odeljcima i Windows API pozivima. Razlika je u tome što sada objekte deklarišemo na sledeći način:

```
var
  Critical1: TCriticalSection;
```

a koristimo ih na sledeći način:

```
// in FormCreate
Critical1 := TCriticalSection.Create;
// in FormDestroy
Critical1.Free;
// in the form loop
Critical1.Enter;
...
Critical1.Leave;
```

Postoji veoma mala razlika, ali je kod nešto lakše napisati. Jedinica SyncObjs deklariše klasu `TCriticalSection` kao i klasu `THandleObject`, klasu `TEvent` i klasu `TSingleEvent`.

Rezime ideja opisanih ovim dugim primerom je da je muteks, kritični odeljak ili semafor neophodan svaki put kada dva procesa pristupaju deljenim podacima ili resursima. U suprotnom, sistem može da prebacuje kontrolu sa procesa na proces pre nego što bilo koji od njih ne završi međuoperaciju. U ovom međustanju podaci mogu biti neispravni, ali će ih ostali procesi ipak koristiti. Ovim objektima sinhronizacije možete pristupiti upotrebom jednostavnih Windows API poziva, ili čak jednostavnijih VCL objekata.

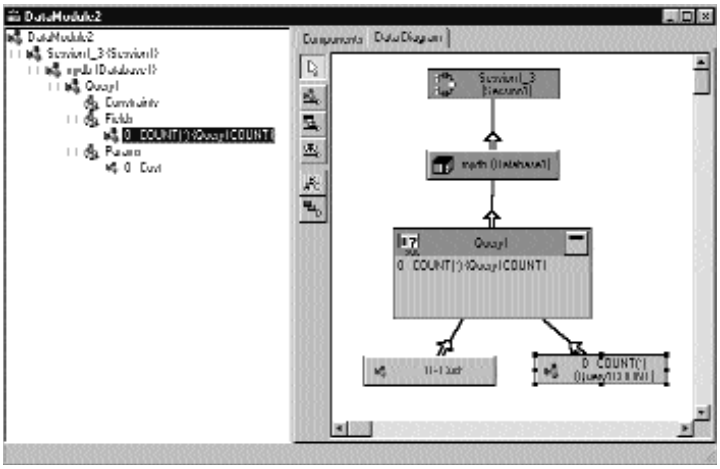
Procesni pristup bazi podataka

Ponekad je zgodno izvršiti odjednom dva različita procesa nad bazom podataka, započinjući odvojeni zahtev (komponentom tabele i upita) pre nego što se izvrši prvi zahtev. Da biste ovo učinili, potrebno je da koristite procese. Ukoliko imate dva upita, na primer, možda želite da izvršite drugi upit pre nego što prvi pokaže svoje rezultate, da biste ubrzali operacije na Vašem (moguće) brzom serveru. U drugim slučajevima, možda želite da izvršite složeni upit ili da pretražite veliku tabelu u okviru procesa, da biste izbegli prekidanje unosa korisnika. Zapravo, kada događaj pokrene upit u glavnom procesu ili operiše nad skupom podataka, korisnik ne može da operiše programom sve dok se ne obavi započeta operacija. Upotreba sekundarnog procesa za izračunavanje bilo čega što odmah nije neophodno, učiniće da aplikacija bude upotrebljivija.

BDE podržava više simultanih zahteva veze jedne aplikacije upotrebom više sesija. Drugim rečima, potrebno je da upotrebimo objekat `TSession` da bismo načinili više veza upotrebom BDE-a iz istog programa (zapravo, jednu vezu iz glavnog programa i jednu iz sekundarnog procesa). Primer ThreadDB pokazuje kako možete da upotrebite proces i zasebne `Session` objekte za obradu u pozadini nad tabelom baze podataka koju trenutno pregledate. Program koristi modul podataka (prikazan na slici 17.6), koji sadrži tri komponente za pristup bazi podataka: bazu podataka, sesiju i upit. Evo odgovarajućeg koda:

```
object Session1: TSession
  Active = True
  AutoSessionName = True
end
object Database1: TDatabase
  AliasName = 'DBDEMOS'
  Connected = True
  DatabaseName = 'mydb'
  SessionName = 'Session1_1'
end
object Query1: TQuery
  DatabaseName = 'mydb'
  SessionName = 'Session1_2'
  SQL.Strings = (
    'select count (*) '
    'from orders'
    'where CustNo = :Cust;' )
  ParamData = <
    item
      Data = ftInteger
      Name = 'Cust'
      Param = ptUnknown
    end>
  object Query1COUNT: TIntegerField
```

```
    FieldName = 'COUNT(*)'
  end
end
```



SLIKA 17.6 *Drvo i dijagram podataka modula podataka primera ThreadDB*

Kakav je cilj aplikacije ThreadDB? Glavni formular ovog primera prikazuje spisak kupaca, a upit modula podataka može da se upotrebi za izračunavanje broja porudžbina odabranog kupca. Kada god se izmeni aktuleni slog tabele kupaca, možda želite da ponovo izračunate broj njegovih porudžbina. Međutim, ukoliko jednostavno izvršite ovaj upit svaki put kada se izmeni slog glavne tabele, program će se sporo izvršavati. Korisnik koji pritiska taster sa strelicom nadole da bi se brzo kretao kroz slogove, moraće da sačeka da se upit izvrši i da se rezultati upita prikažu pre nego što pređe na naredni slog.

Alternativno, program ThreadDB izvršava upit u procesu u pozadini, omogućavajući korisniku da prelazi na novi slog pre nego što upit da svoje vrednosti. Brzim prelaskom na slogove, moći ćete da pokrenete više upita istovremeno, koristeći više instanci modula podataka. Glavni program kreira proces svaki put kada se promeni aktuelni slog tabele Customers:

```
procedure TForm1.Table1AfterScroll (DataSet: TDataSet);
var
  Th1: TDatabaseThread;
begin
  // create and start a new thread
  Th1 := TDatabaseThread.Create (True);
  Th1.Priority := tpLower;
  Th1.FreeOnTerminate := True;
  Th1.CustNo := Table1CustNo.AsInteger;
  Th1.Resume;
end;
```

Primetićete da proces ima nizak prioritet (da bi osetljivost programa na korisničke akcije bila maksimalna) i da prihvata ID korisnika aktuelnog sloga. Čim se započne proces, kreira se novi modul podataka, određuje se parametar sa identifikacijom kupca za upit i upit se otvara:

```
procedure TDatabaseThread. Execute;
begin
  with TDataModule2.Create (nil) do
  begin
    try
      Query1.ParamByName( 'Cust' ).AsInteger := CustNo;
      Query1.Open;
      NewCaption := 'Number of Orders ' +
        Query1Count .AsString;
    finally
      Synchronize (UpdateCaption);
      Query1.Close;
      Free; // the data module
    end;
  end;
end;
```

Izvršavanje ovog programa je prihvatljivo; program omogućava korisniku da promeni aktuelni slog pre nego što se obavi izvršavanje upita u procesu u pozadini. Ipak, ukoliko jednostavno neprekidno držite pritisnut taster sa strelicom nadole neko vreme, započeće se veliki broj simultanih procesa, toliko veliki da Windows i BDE ne mogu da ih obrade. Naravno, nerazumno je započeti neograničeni broj upita, operaciju koja može da naruši performanse bilo kog servera baza podataka. Jednako je nerazumno započeti neograničen broj procesa, jer jezgro operativnog sistema Win32 počinje da se guši već kada je pokrenuto 15 ili 16 procesa.

Iz ovog razloga možemo da omogućimo započinjanje nekoliko procesa, a da se upit jednostavno ne izvrši za druge procese (ili da ih držimo u redu za izvršavanje). Ovu ideju lako možemo da implementiramo upotrebom semafora.

Da bih to učinio, deklarisao sam promenljivu `hSemaphore` tipa `THandle` u jedinici klase procesa, i ovaj kod sam dodao inicijalizacionom odeljku jedinice za kreiranje semafora sa ograničenjem od 10 procesa:

```
initialization
  hSemaphore := CreateSemaphore (
    nil, 10, 10, 'ThDB_MD Semaphore');
```

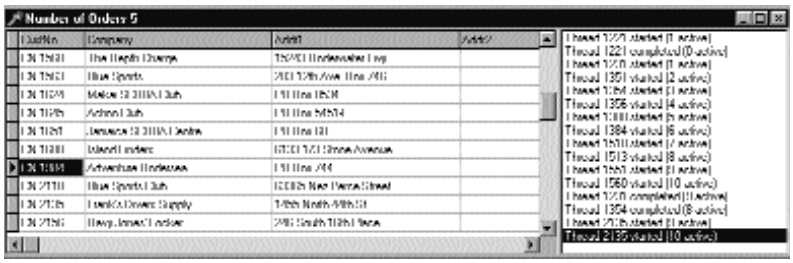
Glavni program koristi ovaj semafor pre nego što pokrene proces, pozivom API funkcije `WaitForSingleObject`. U ovom slučaju funkcija smanjuje brojač semafora ukoliko je veći od nule. Ukoliko je brojač semafora jednak nuli, program čeka da neki proces oslobodi semafor da bi mu pristupio. Čekanje u glavnom programu može dovesti do blokiranja jer procesi moraju da se sinhronizuju sa semaforom (kao što je razmatrano u primeru FindTh). Zbog toga, ukoliko proces nije dostupan, program jednostavno preskače zahtev korisnika:

```
// procedure TForm1.Table1AfterScroll
if WaitForSingleObject (hSemaphore, 0) = Wait_Object_0 then
begin
  // create and start a new thread
  Th1 := TDatabaseThread.Create (True);
  // continue as before
```

Naravno, proces mora pozvati `ReleaseSemaphore` pre nego što se prekine. Evo kompletnog koda, koji takođe beleži aktuelni status procesa u listi glavnog formulara:

```
procedure TDatabaseThread. Execute;
begin
  // log
  Inc (thcount);
  LogText := Format ('Thread %d started (%d active)',
    [CustNo, thcount]);
  Synchronize (AddToLog);

  with TDataModule1.Create (nil) do
  begin
    try
      Query1.ParamByName( 'Cust' ).Aslnteger := CustNo;
      Query1.Open;
      NewCaption := 'Number of Orders ' +
        Query1Count.AsString;
    finally
      Synchronize (UpdateCaption);
      Query1.Close;
      Free; // the data module
      // log
      Dec (thcount);
      LogText := Format ('Thread %d completed (&d active)',
        [CustNo, thcount]);
      Synchronize (AddToLog);
    end;
  end;

  // thread is done, release semaphore
  ReleaseSemaphore (hSemaphore, 1, nil);
end;
```

Sada možete da pokrenete samo 10 procesa istovremeno, dok će drugi zahtevi korisnika biti odbačeni. Program ove zahteve ne čuva u listi, već jednostavno zaboravlja na njih. U svakom slučaju, korisnik je, možda, prešao na naredni slog. Primer izlaza ovog programa možete videti na slici 17.7.



SLIKA 17.7 *U primeru ThDB više procesa u pozadini ažurira naslov formulara (pogledajte dnevnik sa strane)*

Šta je sledeće?

U ovom poglavlju smo razmatrali multitasking, više procesa i sinhronizaciju između procesa i odvojenih procesa. Ovi primeri su demonstrirali da možete da upotrebite Delphi da biste uronili u složeno Windows programiranje upotrebom karakteristika sistema niskog nivoa. Da biste više naučili, pogledajte knjige o Windows API programiranju, a zatim jednostavno primenite te informacije u Delphi programiranju.

Sledeće poglavlje se odnosi na još jednu tehniku koju treba da savladate da biste postali ekspert za Delphi programiranje: to je upotreba debagera i tehnika debagovanja u opšem slučaju. Poglavlje će Vam dati veći uvid u interno funkcionisanje Delphi aplikacija, a takođe će predstaviti i tok Windows poruka.

POGLAVLJE

18

Debagovanje Delphi programa

KADA STE KOMPAJLIRALI PROGRAM U DELPHIJU I POKRENULI GA, MOŽDA STE POMISLILI DA STE ZAVRŠILI POSAO, ALI MOŽDA NISU SVI VAŠI PROBLEMI REŠENI. PROGRAMI MOGU DA IMAJU GREŠKE PRILIKOM IZVRŠAVANJA (RUN-TIME ERRORS), ILI SE MOŽDA NEĆE IZVRŠAVATI ONAKO KAKO STE PLANIRALI. KADA SE NEŠTO OD OVOGA DOGODI, BIĆE POTREBNO DA OTKRIJETE ŠTA JE POŠLO NAOPAKO I KAKO DA TO ISPRAVITE. NA SREĆU, MNOGO OPCIJA I ALATA JE NA RASPOLAGANJU ZA ISTRAŽIVANJE PONAŠANJA WINDOWS APLIKACIJA.

Delphi sadrži ugrađeni debager i nekoliko drugih opcija da bi Vam omogućio da nadgledate rezultat procesa kompilacije na različite načine. Ovo poglavlje sadrži pregled svih ovih tema, demonstrirajući ključne ideje prostim primerima. Prvi deo ovog poglavlja se odnosi na Delphijev integrisani debager i razne karakteristike koje Delphi obezbeđuje za debagovanje prilikom izvršavanja. Zatim ću ja opisati neke tehnike debagovanja i razmatraću kako možete da nadgledate tok poruka u Vašoj aplikaciji. Poslednji deo opisuje kako možete da proverite status memorije koju koristi program.

Upotreba integrisanog debagera

Kao što sam ranije pomenuo, kada pokrenete program iz Delphi okruženja, interni debager zapravo izvršava program. (Ovo ponašanje možete da promenite onemogućavanjem opcije Integrated Debugger u okviru za dijalog Debbager Options.) Većina Run komandi se odnosi na debager. Neke od ovih komandi su takođe dostupne iz podmenija Debug, Editorovog kontekst menija.

Kada se program izvršava u debageru, kada kliknete kontrolu Pause na SpeedBaru, suspendovaćete izvršavanje. Kada se program suspenduje i kada kliknete kontrolu Step Over, program se izvršava korak po korak. Takođe, program možete da izvršavate korak po korak od samog početka ukoliko kliknete kontrolu Step Over dok se nalazite u modu za dizajniranje. Ipak, uzmite u obzir da su Windows aplikacije vođene događajima, te zaista ne postoji način da aplikaciju izvršavate korak po korak, kao što možete da učinite sa DOS aplikacijama. Zbog toga, najuobičajeniji način da debagujete Delphi aplikaciju (ili bilo koju drugu aplikaciju) jeste da odredite tačke prekida (*breakpoint*) u delu koda koji želite da debagujete.

Kada je program zaustavljen u debageru, možete nastaviti sa izvršavanjem programa upotrebom komande Run. Ovim ćete program zaustaviti na sledećoj tački prekida. Alternativno, možete mnogo bliže nadgledati izvršavanje praćenjem (tracing) programa. Možete korstiti komandu Step Over (taster F8) da biste izvršili narednu liniju koda, komandu Trace Into (taster F7) da biste ušli u izvorni kod funkcije ili metoda (to jest, da biste izvršili kod podrutina korak po korak i da biste izvršili kod podrutina koje se pozivaju iz podrutina, i tako dalje). Delphi ističe liniju koja treba da se izvrši drugačijom bojom i malom ikonom u obliku strelice, tako da možete videti šta Vaš program radi.

Treća opcija, Trace to Next Source Line (Shift + F7), će prebaciti kontrolu na narednu liniju izvornog koda Vašeg programa koji se izvršava, bez obzira na tok. Ova naredna linija može biti naredna linija izvornog koda (kao kod komande Step Over), linija unutar funkcije koju poziva Vaš program (kao kod komande Trace Into) ili linija koda unutar obrade događaja ili funkcije programa koju aktivira sistem. Ukoliko želite da nadgledate efekat izvršavanja zadate linije koda, takođe se možete pomeriti na tu poziciju i pozvati komandu Run to Cursor (taster F4). Program će se izvršavati sve dok se ne dođe do te linije, tako da je ovo slično određivanju privremene tačke prekida. Konačno, nova Delphi 5 komanda Run until Return (Shift + F8) izvršava metod ili funkciju sve dok se ona ne završi. Ovo je zgodna mogućnost kada počnete da pratite funkciju koju ne želite da debagujete.

Debagovanje biblioteka (i ActiveX kontrola)

Takođe možete da upotrebite integrisani debager za debagovanje DLL-a ili bilo koje druge vrste biblioteke (kao što je ActiveX kontrola). Jednostavno otvorite Delphi projekat biblioteke,

odaberite komandu menija Run➥Parameters i unesite naziv za Host Application. (Opcija je aktivna samo ukoliko je odredište aktuelnog projekta biblioteka.) Sada kada kliknete kontrolu Run (ili taster F9), Delphi će pokrenuti glavni izvršni fajl, koji će zatim učitati biblioteku. Ukoliko odredite tačku prekida unutar izvornog koda biblioteke, izvršavanje će se prekinuti unutar koda biblioteke, kao što se i očekuje.

Slično, možete da upotrebite ovu mogućnost za debagovanje ActiveForma. Jednostavno unesite putanju web pretraživača za Host Application, na primer `C:\Program Files\Microsoft Internet\Iexplore.exe`; zatim unesite putanju HTML test fajla kao parametar Run. Da biste obavili ovaj posao, trebalo bi takođe da upotrebite komandu menija Run➥Register ActiveX Server. Kada je ActiveX registrovan, web pretraživač će koristiti tu verziju, a ne neku drugu verziju koja je dostupna u OCX kešu.

Informacije debagovanja

Da biste debagovali Delphi program, morate da dodate informacije o debagovanju Vašem kompajliranom kodu. Delphi ovo čini po definiciji, ali ukoliko je ovo isključeno, možete ponovo uključiti informacije upotrebom okvira za dijalog Project Options. Kao što možete videti na slici 18.1, strana Compiler sadrži odeljak Debugging sa četiri polja za potvrdu:

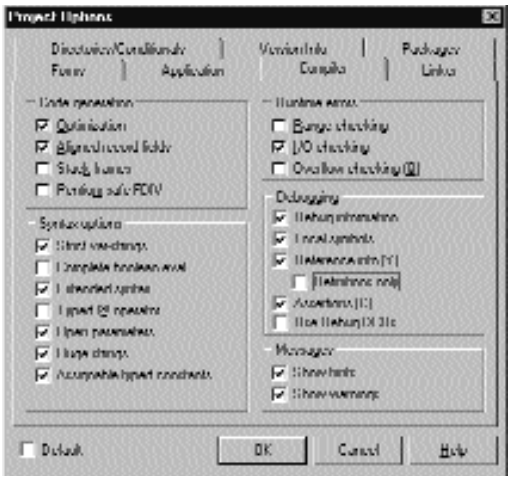
- **Debug Information** dodaje svakoj jedinici mapu izvršnih adresa i odgovarajuće brojeve linija izvornog koda. Na ovaj način se povećava veličina DCU fajla, ali nema uticaja na veličinu i brzinu izvršnog programa. (Linker ne uključuje ovu informaciju kada izrađuje EXE fajl, izuzev ukoliko eksplicitno ne zahtevate TD32 informaciju debagovanja, koja je u drugačijem formatu.)
- **Local Symbols** dodaje informaciju debagovanja o svim lokalnim identifikatorima, nazive i tipove simbola u implementacionom odeljku jedinice.
- **Reference Info** dodaje informacije o referencama o simbolima definisanim u modulu da bi se omogućilo Project Inspectoru (ili Object Browseru) da ih prikaže. Ukoliko nije potvrđena podopcija **Definitions Only**, informacije će uključiti svaku upotrebu simbola, omogućavajući unakrsne reference u Project Exploreru.
- **Assertions** Vam omogućava da dodate alegacije, kod koji će zaustaviti Vaš program ukoliko određeni test ne uspe. Za razliku od izuzetaka ili drugog koda za detekciju greške, alegacije se automatski mogu ukloniti iz Vašeg programa; jednostavno isključite ovu opciju. Alegacije ću razmatrati kasnije u ovom poglavlju. Posle ove izmene, ponovo ćete morati da izradite Vaš projekat da biste dodali ili uklonili kod alegacija iz Vaše aplikacije.
- **Use Debug DCUs** (koristite Debug DCUs) da biste povezali debagovanu verziju DCU fajlova VCL-a u Vaš program. U praksi, ova opcija dodaje putanju Debug DCU (naznačenu u strani General okvira za dijalog Debugger Options) putanji Search (naznačenoj na strani Directories/Conditionals u okviru za dijalog Project Options).

Integrisani debager koristi ove informacije prilikom debagovanja programa. Informacije debagovanja se ne pridružuju izvršnom fajlu izuzev ukoliko ne odaberete opciju TD32 Debug Info na strani Linker okvira za dijalog Project Options. Informacije o debagovanju bi trebalo da dodate Vašem izvršnom fajlu samo ukoliko planirate da koristite spoljašnji debager, recimo, Borlandov

Turbo Debugger for Windows (TD32). Nemojte uključivati ove informacije ukoliko planirate da koristite samo integrisani debager, i ne zaboravite da ih uklonite iz izvršnog fajla koji prosleđujete.

Udaljeno debagovanje

Mogućnost koja je prvo predstavljena u Delphiju 4 je udaljeno debagovanje. Ova tehnika Vam omogućava da debagujete program koji se izvršava na drugom kompjuteru, obično na serveru. Da biste aktivirali udaljeno debagovanje, najpre morate da instalirate klijenta udaljenog debagovanja na odredišnu mašinu. Zatim je potrebno da pokrenete klijenta udaljenog debagovanja komandom `borrdg.exe -listen`, po mogućstvu ga pokrećući kao servis u Windowsu NT.



SLIKA 18.1 *Upotrebite stranu Compiler okvira za dijalog Project Options da biste uključili informacije o debagovanju u kompajliranu jedinicu*

Sada je potrebno da kompajlirate Vaš program uključujući simbole udaljenog debagovanja na strani Linker okvira za dijalog Project Options. Takođe, možete podesiti direktorijum Output na udaljenu mašinu na strani Directories/Conditionals istog okvira za dijalog, tako da ne morate ručno da kopirate program i odgovarajući RMS fajl na udaljeni kompjuter svaki put kada ponovo kompajlirate program. Konačno, odredite udaljenu putanju i projekat pod Run➥Parameters, popunjavajući polje Remote nazivom mašine ili IP adresom.

Kada je sve pravilno podešeno, moći ćete da koristite Delphijev integrisani debager za debagovanje programa koji se izvršava na udaljenom kompjuteru. Imaćete mogućnost da odredite tačke prekida i izvršite sve standardne operacije debagovanja kao i obično.

Attach to Process

Novo u Delphiju 5 je karakteristika Attach to Process koja je dostupna preko komande Run. Ova nova karakteristika omogućava da započnete debagovanje programa koji se već izvršava na sistemu. Na primer, možda želite da pridružite debager procesu koji je upravo prikazao izuzetak da biste shvatili šta je pošlo naopako.

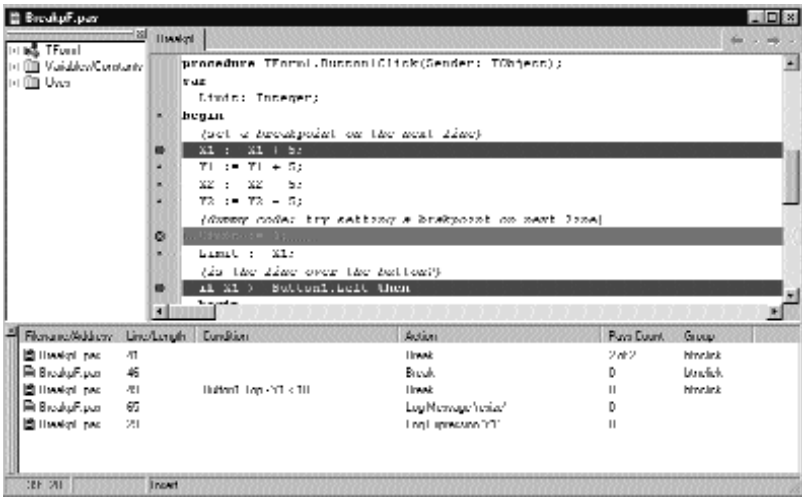
Kada odaberete komandu Attach to Process, prikazaće se lista procesa koji se izvršavaju. Ukoliko odaberete Delphi program za koji imate izvorni kod, imaćete ponovo tradicionalnu situaciju debagovanja. Ukoliko odaberete neki drugi program za koji nemate izvorni kod, moći ćete samo da pratite njegovo izvršavanje na asemblerskom nivou, koristeći CPU prozor, ali ne i editor izvornog koda.

Upotreba tačaka prekida

Postoji nekoliko tipova tačaka prekida u Delphiju:

- **Tačke prekida izvornog koda** (source breakpoints) i **tačke prekida adresa** (adress breakpoints) su slične jer zaustavljaju izvršavanje kada se procesor sprema da izvrši instrukcije na određenoj adresi.
- **Tačke prekida podataka** (data breakpoints) zaustavljaju izvršavanje kada se promeni vrednost na zadatoj lokaciji.
- **Tačke prekida učitavanja modula** (module load breakpoints) zaustavljaju izvršavanje kada se učita određeni modul koda.

Kao što ime nagoveštava, kada se dostigne tačka prekida, ona treba da zaustavi izvršavanje programa. U Delphiju 5 tačke prekida mogu učiniti i više od samog zaustavljanja — svaka tačka prekida može imati jednu od nekoliko akcija koje su joj pridružene. Ove akcije mogu biti tradicionalna tačka prekida, prikazivanje fiksnog stringa ili izraz koji se izračunava u dnevniku poruka, ili aktiviranje ili deaktiviranje drugih grupa tačaka prekida. Prozor Breakpoint List (videti sliku 18.2) prikazuje ove dodatne informacije, kao i opis tačaka programa celog programa. Slika je preuzeta iz jednostavnog programa, programa BreakP, koji ću koristiti da bih ilustrovao neke od karakteristika tačaka prekida u Delphiju.



SLIKA 18.2 *Prozor Breakpoint List kada prikazuje uslovnu tačku prekida i kada je dokiran na dnu editora*

Druga nova karakteristika je da tačke prekida možete dodeliti grupama. Zatim možete aktivirati ili deaktivirati sve tačke prekida grupe odjednom, bilo upotrebom direktne komande (iz kontekst menija prozora Breakpoint List) bilo kao automatski efekat akcije tačke prekida.

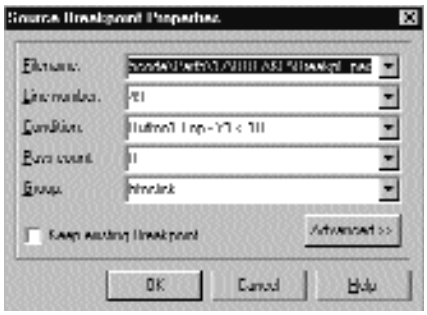
Tačke prekida izvornog koda

Ukoliko želite prekid prilikom izvršavanja iskaza u Vašem izvornom kodu, upotrebićete tačke prekida izvornog koda, koje su inače najčešće tačke prekida. Tačku prekida izvornog koda možete kreirati kada kliknete tanku liniju u prozoru editora koda, kada kliknete desnim tasterom miša određenu liniju izvornog koda ili ukoliko odaberete komandu Toggle Breakpoint iz kontekst menija, ili upotrebom okvira za dijalog Add Source Breakpoint. (Ovaj okvir za dijalog možete prikazati ukoliko odaberete komandu menija Run➥Add Breakpoint➥Source Breakpoint, ukoliko odaberete Add➥Source Breakpoint iz lokalnog menija prozora Breakpoint List, ili ukoliko pritisnete taster F5.)

Kada kreirate novu tačku prekida izvornog koda (upotrebom bilo koje od ovih tehnika), prikazaće se ikona na levoj margini koda, a linija izvornog koda će biti prikazana drugom bojom. Ipak, ne možete odrediti valjanu tačku prekida na bilo kojoj liniji koda. Tačka prekida izvornog koda je valjana samo ukoliko je Delphi za tu liniju generisao izvršni kod. To znači da ne možete tačkom prekida označiti komentar, deklaraciju, liniju direktive kompajlera, ili bilo koju drugu liniju koja nije izvršna. Ukoliko ste kompajlirali program bar jednom (kada su uključene informacije o debagovanju), tačkice u okviru na levoj strani označavaju izvorne linije na koje ste smestili valjane tačke prekida. Mada možete da odredite tačku prekida na poziciji koja nije valjana, Delphi će Vas upozoriti kada pokrene program i označiće tačku prekida koja nije valjana drugačijom ikonom i drugačijom bojom.

Takođe, imajte na umu da — pošto koristi kompajler koji vrši optimizaciju — Delphi neće generisati bilo kakav izvršni kod za linije izvornog koda do kojih se nikada neće doći u Vašem programu, niti za bilo koju drugu liniju koja ne utiče na logiku programa. Ukoliko kreirate tačku prekida izvornog koda koja nije valjana, a zatim izvršavate program korak po korak, debager će preskočiti liniju jer taj kod ne postoji u optimizovanoj verziji kompajliranog programa. Postoji primer tačke prekida koja nije valjana u jedinici BreakF programa BreakP.

Kada ste odredili valjanu tačku prekida izvornog koda, možete promeniti neka od njenih svojstava. Okvir za dijalog Source Breakpoint Properties je dostupan iz prozora sa listama Breakpoint ukoliko desnim tasterom miša kliknete ikonu u editoru. U ovom prozoru (videti sliku 18.3) možete da odredite uslove za tačku prekida, naznačite njen brojač prosleđivanja i dodelite je grupi. Takođe možete kliknuti kontrolu Advanced da biste prikazali proširenu verziju prozora, koja nudi akcije tačke prekida koje su predstavljene u Delphiju 5, a koje ću razmatrati u narednom odeljku.



SLIKA 18.3 *Standardni deo okvira za dijalog Source Breakpoint Properties*

Na slici 18.3 možete videti definiciju *uslovne* (conditional) tačke prekida, koja se koristi da prekine program samo kada je zadovoljen određeni izraz. Na primer, možda želimo da zaustavimo izvršavanje metoda `Button1Click` u primeru BreakP samo kada su se linije (naznačene promenljivom Y1) pomerile blizu kontrole, upotrebom sledećeg uslova:

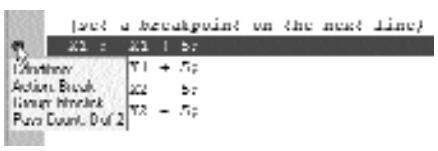
```
Button1.Top - Y1 < 10
```

Uslov se takođe dodaje prozoru sa listama Breakpoint kao što možete videti na slici 8.2. Sada možete da pokrenete program i kliknete kontrolu više puta. Tačka prekida se ignoriše sve dok se ne ispuni uslov (kada je `Button1.Top - Y1` manje od 10). Samo kada ste kliknuli kontrolu nekoliko puta, program će se zaustaviti u debageru.

Ukoliko znate koliko puta treba dopustiti da se izvrši linija koda pre nego što debager treba da zaustavi program, možete da odredite vrednost za Pass Count. Čim debager dođe do tačke prekida, povećaće brojač sve dok ne dostigne broj prolaza. Prozor sa listama Breakpoint će naznačiti status (videti prvu liniju slike 18.2), prikazujući "2 od 5" ukoliko je program izvršio liniju dva puta pošto ste odredili vrednost 5.

Polje za potvrdu Keep Existing Breakpoint prozora Breakpoint Properties se koristi kada želite da duplirate postojeću tačku pomeranjem linije izvronog koda na koju se referiše tačka. Ukoliko opcija nije potvrđena, postojeća tačka prekida se pomera; ukoliko je opcija potvrđena, kreira se nova tačka prekida. Ovim se obezbeđuje vrsta mehanizma za kloniranje ukoliko želite da kreirate novu tačku prekida na osnovu postojeće.

Konačno, primetićete da su u Delphiju 5 sve informacije koje se prikazuju u prozoru sa listama Breakpoint na raspolaganju i u oblačiću kada pomerite pokazivač miša iznad ikone tačke prekida u editoru, kao što je pokazano na slici 18.4.



SLIKA 18.4 *Novi oblačić za tačke prekida*

SAVET

Tačke prekida koje dodajete programu, čuvaju se u projekt fajlu radne površine (`*.dsk`) ukoliko je omogućeno Desktop Saving. Čuvanjem ove informacije moći ćete ponovo da otvorite projekat i ponovo započnete sesiju debagovanja. Takođe, možete da zadržite postojeće tačke prekida, bar one najsloženije, za buduću upotrebu: jednostavno ih deaktivirajte i sačuvajte podešavanja radne površine umesto da uklonite tačke prekida. ■

Akcije tačaka prekida

Kao što sam ranije pomenuo, Delphi 5 čini tačke prekida nešto fleksibilnijim. Pored prostog zaustavljanja izvršavanja programa, one mogu da izvrše i druge akcije, kao što je prikazano proširenom verzijom prozora Breakpoint Properties (do kojeg možete doći ukoliko kliknete kontrolu Advanced), koji je prikazan na slici 18.5.

SLIKA 18.5 *Napredna verzija prozora Breakpoint Properties*

Ukoliko onemogućite polje za potvrdu Break, program se neće zaustaviti kada debager dođe do te linije koda. Možete da zatražite prekid naredni put kada se dođe do tačke prekida, ili jednostavno pošaljite poruku ili rezultat izraza prozorima Event (koje ću kasnije opisati), kao što je pokazano na ilustraciji. Takođe možete da aktivirate ili deaktivirate tačke prekida samo ukoliko je ispunjena uslovna tačka prekida.

Možda se pitate kada je upisivanje poruke u dnevnik bolje nego efektivno zaustavljanje programa. Postoje neki interesantni slučajevi, kao što to pokazuje program BreakP. Jedna od opcija dnevnika poruka se koristi u obradi događaja formulara `OnResize`. Ukoliko prevučete borduru formulara, ovaj događaj će se pozvati nekoliko puta zaredom, svaki put prekidajući debager.

Ovaj problem je još očigledniji kod obrade događaja `OnPaint`. Ukoliko se prozori editora i formulara preklapaju, ući ćete u beskonačnu seriju tačaka prekida. Svaki put kada se formular ponovo iscrta, tačka prekida zaustavlja program, pomerajući prozor editora u prvi plan ispred

formulara i uslovljavajući ponovno iscrtavanje formulara, što zaustavlja program na istoj tački prekida — iznova i iznova. Pokušajte da prozore editora i formulara postavite tako da se ne preklapaju. Daleko pametnije rešenje je da upotrebite uslovnu tačku prekida, odredite broj prolaza ili zapišete poruku u dnevnik sa vrednošću za koju ste zainteresovani, kao što je promenljiva `Y1` u slučaju koji je prikazan na slici 18.5.

Takođe, Windows poruke promene fokusa su veoma složene za debagovanje upotrebom tačaka prekida, jer će prelazak u debager za tačku prekida prouzrokovati da program koji debagujete izgubi fokus. Ovo ne možete zaobići promenom položaja prozora tako da se ne preklapaju, kao što je slučaj za događaj `OnPaint`; Vaš izbor je ili zapisivanje poruka u dnevnik ili udaljeno debagovanje. Udaljeno debagovanje je odličan alat za "neinvaziono" debagovanje problema kao što je iscrtavanje ili poruka promene fokusa.

Tačke prekida adresa

Ukoliko nemate izvorni kod za neku proceduru ili funkciju, želećete da kreirate tačku prekida adrese da biste zaustavili izvršavanje programa na zadatoj tački. Ipak, bez izvornog koda koji će Delphi koristiti u izračunavanju adrese gde želite da načinite pauzu, potrebno je da upotrebite tehnike za određivanje adrese koda koja je u pitanju. Možete da kreirate tačku prekida adrese direktno u CPU pogledu (koji ćemo razmatrati kasnije u ovom poglavlju) ili indirektno (kada imate adresu) upotrebom okvira za dijalog Add Address Breakpoint.

Da biste kreirali tačku prekida adrese iz CPU pogleda, jednostavno kliknite liniju na Disassembly Paneu, pored instrukcije gde želite da načinite pauzu, ili kliknite desnim tasterom miša instrukciju i odaberite Toggle Breakpoint iz lokalnog menija. Kada ste kreirali tačku prekida adrese u izvrornom kodu editora, možete promeniti njena svojstva tako što ćete kliknuti desnim tasterom miša ikonu tačke prekida (ne instrukciju) i odabrati Breakpoint Properties iz lokalnog menija.

Da biste indirektno kreirali tačku prekida adrese, potrebno je da odredite adresu instrukcije gde želite da načinite pauzu u izvršavanju. Ukoliko nameravate da zaustavite izvršavanje programa, a izvorni kod nije deo Vašeg projekta (kao kod standardnih VCL metoda), potrebno je da odredite adresu funkcije ili procedure koja je već kompajlirana.

Jedan način za određivanje adrese metoda objekta je da upotrebite Debug Inspector za taj objekat u vreme izvršavanja. Kasnije u ovom poglavlju ćemo se detaljno upoznati sa Debug Inspectorom. Za sada ćemo razmotriti samo način dobijanja adrese metoda za koji nemate izvorni kod. Na primer, ukoliko želite da načinite pauzu kada korisnik klikne kontrolu, ali pre nego što se uđe u kod obrade događaja, možete potražiti metod `TButton.Click`. Da biste to učinili, kliknite desnim tasterom miša deklaraciju kontrole u deklaraciji klase formulara i odaberite Debug➥Inspect. U prozoru za kontrolu Debug Inspector kliknite karticu Methods, pređite do samog dna liste i pronađite metod `StdCtrls.TButton.Click`.

Pored naziva metoda videćete heksadecimalnu adresu u zagradama. Kopirajte ovu adresu (uključujući i prefiks $), a zatim kreirajte novu tačku prekida adrese koristeći tu vrednost. Kada nastavite sa izvršavanjem programa i kliknete kontrolu, prikazaće se prozor CPU View i u njemu ćete videti disasemblirane instrukcije za metod `TButton.Click`.

SAVET

Izuzev ukoliko ne posedujete Standard verziju Delphija, imate VCL izvorni kod. Trebalo bi da znate da je jedan od njegovih glavnih zadataka u debagovanju programa. Možete uključiti izvorni kod biblioteke u Vaš program i upotrebiti debager da biste pratili njegovo izvršavanje. Naravno, trebalo bi da budete dovoljno hrabri i da imate dovoljno slobodnog vremena da se udubite u složenost VCL izvornog koda. Ali, kada izgleda da ništa drugo ne pomaže, ovo može biti jedino rešenje. Da biste uključili izvorni kod biblioteke, jednostavno dodajte naziv direktorijuma sa VCL izvornim kodom (po definiciji je to `Source\VCL` pod Vašim Delphi direktorijumom) combo polju Search Path strane Directories/Conditionals okvira za dijalog Project Options. Alternativa je da povežete Debug DCU-e, kao što je to opisano ranije u ovom poglavlju. Zatim ponovo izradite ceo program i započnite debagovanje. Kada dođete do iskaza koji sadrže pozive metoda i svojstava, možete pratiti program i videti kako se VCL kod izvršava liniju po liniju. Naravno, možete nad VCL kodom učiniti bilo koju operaciju debagovanja koju možete učiniti i sa svojim kodom. ■

Tačke prekida podataka

Tačke prekida podataka su dramatično različite od tačaka prekida izvornog koda i tačaka prekida adresa jer nadgledaju memorijsku adresu u očekivanju promene vrednosti. Nema nikakve veze gde se nalazi kod koji menja podatke u toj memorijskoj lokaciji; debager će momentalno zaustaviti program i prikazati tačku izvršavanja. Prikaz će biti ili u prozoru editora izvornog koda, ukoliko je izvorni kod deo projekta, ili u CPU View prozoru ukoliko izvorni kod nije na raspolaganju.

Tačke prekida podataka možete odrediti na dva načina. Prva tehnika uključuje zaustavljanje izvršavanja upotrebom tačke prekida izvornog koda na mestu u izvornom kodu gde je identifkator kojeg želite da pratite u opsegu. Kada je program zaustavljen, možete direktno uneti naziv identifikatora u polje Address okvira za dijalog Add Data Breakpoint, a Delphi će izračunati adresu promenljive.

Drugi način za određivanje tačke prekida podataka je da prvo kreirate stražu za promenljivu, što ćemo razmatrati kasnije u ovom poglavlju. Zatim, zaustavićete izvršavanje kreiranjem tačke prekida izvornog koda na mestu gde je identifikator u opsegu. Kada prikažete prozor Watch List, kliknite desnim tasterom miša stražu identifikatora i odaberite Break when Changed iz lokalnog menija. Bilo kojim od ovih metoda, bilo kakva promena u promenljivoj će zaustaviti program i prikazati aktuelnu tačku izvršavanja.

Tačke prekida učitavanja modula

Ukoliko želite da zaustavite izvršavanje kada se učita određeni modul koda, kreiraćete tačku prekida učitavanja modula. Tačku prekida učitavanja modula možete kreirati kada odaberete Run➥Add Breakpoint➥Module Load Breakpoint i kada zatim odaberete EXE ili DLL koji želite da nadgledate. Kada se taj modul učita, Delphi će suspendovati izvršavanje programa i istaknuće tačku izvršavanja, bilo u prozoru editora izvornog koda bilo u prozoru CPU View.

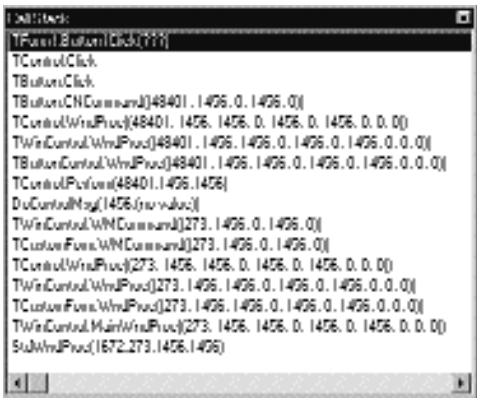
Lakši način da postignete isti efekat je da otvorite prozor Module, pokrenete program do tačke prekida izvrornog koda koja se dešava posle učitavanja svih modula i da zatim odaberete module na čije učitavanje želite da postavite tačku prekida. Možete odrediti tačku prekida učitavanja modula za dati modul ukoliko kliknete modul desnim tasterom miša i odaberte Break on Load iz lokalnog menija, ili ukoliko kliknete liniju liste modula u prozoru Modules.

Pogledi debagera

Dok debagujete program, postoji mnogo prozora (ili pogleda) koje možete otvoriti da biste nadgledali izvršavanje programa i njegov status. Većina ovih prozora je prilično intuitivna, te ću ih ja samo ukratko predstaviti i navešću nekoliko saveta. Da biste ih aktivirali, upotrebite Debug menija View, Delphi IDE-a.

Call stack

Dok pratite program, možete prikazati niz poziva podrutina koje se trenutno nalaze na steku. Informacije steka poziva (call stack) su naročito korisne kada imate veliki broj ugnežđenih poziva ili kada koristite Debug DCUs VCL-a. Drugi primer upotrebe je prikazan na slici 18.6 za jednostavnu obradu događaja `OnClick`, koja se poziva posle mnogih internih VCL poziva.



SLIKA 18.6 *Prozor Call Stack kada je kliknuta kontrola (kada je aktivirana opcija Debug DCUs)*

Prozor Call Stack prikazuje nazive metoda koji se nalaze na steku i parametre koji su prosleđeni svakom pozivu funkcije. Na vrhu prozora je prikazana poslednja funkcija koju je pozvao Vaš program, za kojom sledi funkcija koja ju je pozvala i tako dalje. Na slici možete videti da obradu događaja `Button1Click` klase `TForm1` poziva metod `Click` klase `TControl`, koji poziva isti metod izvedene klase `TButton`, koji je odmah aktiviran porukom metoda `WndProc` koji vrši obradu. Postoji više poziva metoda `WndProc` jer je on ponovo definisan u mnogim VCL klasama.

NAPOMENA

Ukoliko ste zainteresovani, kompletan tehnički opis ovih koraka iz Windows poruka u Delphi obradu događaja možete pronaći u knjizi "Delphi Developer's handbook" (Sybex, 1998). ■

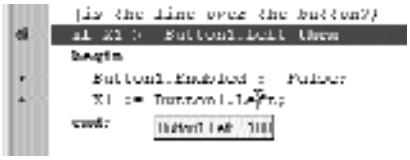
Proveravanje vrednosti

Kada je program zaustavljen u debageru, možete proveriti vrednost bilo kog identifikatora (promenljivih, objekata, komponenata, svojstava i tako dalje) kojima se može pristupiti iz trenutne tačke izvršavanja (to jest, samo identifikatorima koji se trenutno nalaze u opsegu).

Postoje mnogi načini da ovo postignete: upotrebom fly-by oblačića debagera, upotrebom okvira za dijalog Evaluate/Modify, dodavanjem straže u Watch List, ili upotrebom prozora Local Variables ili Debug Inspectora.

Fly-by debager oblačići

Kada je Delphi 3 predstavio *fly-by oblačiće* (fly-by evaluation hints), ova karakteristika je odmah postala jedan od najuobičajenijih načina za proveravanje vrednosti u vreme izvršavanja. Dok je program zaustavljen u debageru, možete pomeriti pokazivač miša iznad promenljive, objekta, svojstva, polja ili bilo kog drugog identifikatora koji označava vrednost, i momentalno ćete dobiti oblačić koji će prikazati trenutnu vrednost identifikatora, kao što možete videti na slici 18.7.



SLIKA 18.7 *Jedna od najkorisnijih karakteristika debagovanja: fly-by oblačići*

Za proste promenljive, kao što su `X1` ili `Y1` u primeru BreakP, i za svojstva objekta (kao na slici) fly-by oblačići prikazuju odgovarajuću vrednost, koju je lako razumeti. Međutim, šta se dešava kada je u pitanju objekat kakav je npr. `Form1` ili `Button1`? Prethodne verzije Delphija su koristile minimalistički pristup, prikazujući samo njegova privatna polja. Delphi 5 prikazuje celokupan skup svojstava objekta, kao što možete videti na slici 18.8. Ovo je poboljšanje, ali smatram da upotreba Debug Inspectora (pogledajte naredna poglavlja) čini status objekta čitljivijim.

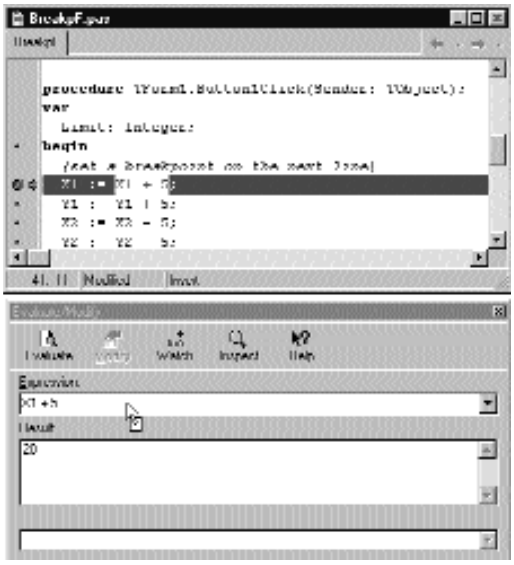
SLIKA 18.8 *Fly-by oblačić za objekat u Delphiju 5*

Zapamtite da možete da prikažete vrednost promenljive kada je program zaustavljen u debageru, ali ne i kada se izvršava. Uz to, možete proveriti samo promenljive koje su vidljive u trenutnom opsegu, jer moraju postojati da biste mogli da ih vidite!

Prozor Evaluate/Modify

Okvir za dijalog Evaluate/Modify se još uvek može koristiti za prikazivanje vrednosti složenog izraza i za modifikovanje vrednosti promenljive ili svojstva. Najlakši način da otvorite ovaj okvir za dijalog jeste da odaberete promenljivu u editoru koda i da zatim odaberete Evaluate/Modify iz kontekst menija editora (ili da pritisnete kombinaciju tastera Ctrl + F7). Duge selekcije se ne

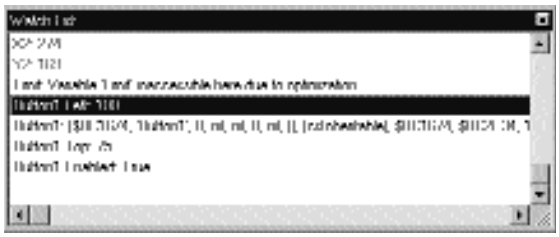
koriste automatski; da biste selektovali dugačak izraz, najbolje ga je kopirati iz editora i smestiti ga u okvir za dijalog. U Delphiju 5 sada takođe možete prevući promenljivu ili ceo izraz iz editora izvornog koda u okvir za dijalog Evaluate/Modify (videti sliku 18.9).



SLIKA 18.9 *Okvir za dijalog Evaluate/Modify se može koristiti za proveru (i promenu) vrednosti promenljive. Takođe možete prevući izraz iz editora u ovaj prozor*

Prozor Watch List

Kada želite da iznova i iznova testirate vrednost grupe promenljivih, upotreba fly-by oblačića može postati malo zamorna. Kao alternativu možete odrediti neke straže (watches — elemente liste promenljivih za koje ste zainteresovani) za promenljive, svojstva ili komponente. Na primer, možete postaviti stražu za svaku vrednost koja se koristi u događaju `Button1Click` primera BreakP, koja se poziva svaki put kada korisnik klikne kontrolu. Ja sam dodao veliki broj straža da bih prikazao vrednosti najrelevantnijih promenljivih i svojstava koji su uključeni u ovaj metod, kao što možete videti na slici 18.10. Kao što je ranije istaknuto, ovaj prozor možete takođe koristiti kao početno mesto za određivanje tačaka prekida.



SLIKA 18.10 *Upotreba prozora Watch List*

Straže možete odrediti upotrebom komande Add Watch at Cursor lokalnog menija editora (ili tako što ćete da pritisnete kombinaciju tastera Ctrl + F5), ali brža tehnika u Delphiju 5 je da prevučete promenljivu ili izraz iz izvornog koda u prozor Watch List. Kada dodate stražu, biće potrebno da odaberete odgovarajući format za izlaz, a možda će biti potrebno da unesete tekst za složenije izraze. Ovo se postiže kada dva puta kliknete stražu u listi, čime se otvara okvir za dijalog Watch Properties, ili ukoliko upotrebite ekvivalentnu komandu Edit Watch iz kontekst menija.

SAVET

Imajte na umu da se ovaj prozor, kao i mnogi drugi prozori za debagovanje, može držati vidljivim dokiranjem uz editor ili upotrebom opcije Stay on Top. ■

Prozor Local Variables

Još jedna korisna karakteristika Delphija je prozor Local Variables (prozor lokalnih promenljivih). Ovaj prozor automatski prikazuje naziv i vrednost bilo koje lokalne promenljive u trenutnoj proceduri ili funkciji kada se program zaustavi u tački prekida. Za metode ćete takođe videti implicitne privatne podatke promenljive `Self`. Prozor Local Variables je veoma slaličan prozoru Watch List, ali ne možete podesiti njegov sadržaj, jer se automatski ažurira kada pređete na praćenje nove funkcije ili metoda, ili se zaustavite u nekoj drugoj tački prekida.

Za bilo koju referencu objekta koja se pojavljuje u prozoru Local Variables (ili prozoru Watch List), kao i za prikazivanje njegovih detaljnih vrednosti u jednoj liniji, takođe možete otvoriti prozore Debug Inspector. Da biste to učinili, dva puta kliknite promenljivu u prozoru Local Variables ili upotrebite komandu Inspect kontekst menija prozora Watch List.

Debug Inspector

Prozori Debug Inspector Vam omogućavaju da pogledate podatke, metode i svojstva objekta ili komponente u vreme izvršavanja, a korisnički interfjes je veoma sličan Object Inspectoru u vreme dizajniranja (kao što možete videti na slici 18.11). Osnovna razlika je u tome da Debug Inspector ne prikazuje samo published svojstva, već celu listu svojstava, metoda i lokalnih polja podataka objekta, uključujući i privatna. Kao što je već opisano, da biste aktivirali sličan Inspector u vreme debagovanja, možete odabrati vidljiv identifikator u editoru, aktivirati lokalni meni i odabrati Debug➥Inspect.

SLIKA 18.11 *Prozor Debug Inspector koji prikazuje svojstva kontrole*

SAVET

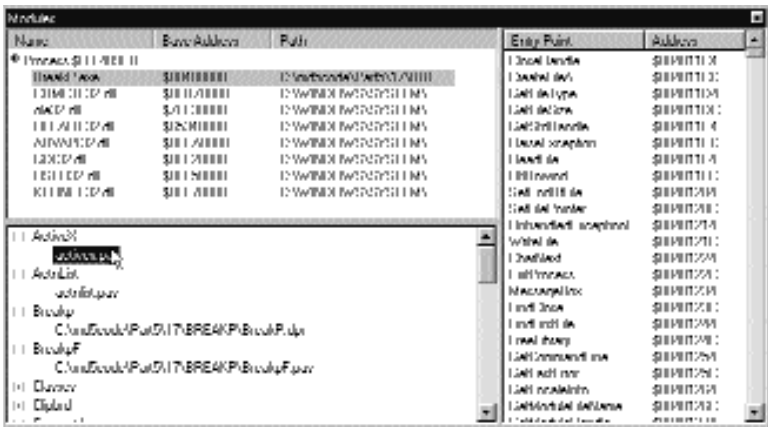
Debug Inspector je sličan komponenti Object Debugger koju sam napisao za "Delphi Developer's Handbook" i dostupan je na mom web sajtu (`www.marcocantu.com`). Ova komponenta Vam omogućava da dobijete potpunu listu vrednosti published svojstva komponente u vreme izvršavanja. ■

Videćete da Debug Inspector prikazuje definiciju svojstava a ne njihove vrednosti. Da biste ovo aktivirali, potrebno je da selektujete svojstvo i kliknete malu kontrolu označenu znakom pitanja na desnoj strani. Na ovaj način se izračunava vrednost, ukoliko je dostupna. Takođe, možete izmeniti podatke objekata ili vrednost svojstva.

Ukoliko proveravate parametar `Sender`, takođe možete konvertovati ceo objekat u neki drugi tip, tako da možete videti njegova specifična svojstva (bez konverzije dobićete samo informacije o generičkoj strukturi `TObject`). Kada radite sa komponentom, možete proveriti podobjekat, recimo font. Možete upotrebiti više Debug Inspector prozora ili koristiti samo jedan i vratiti se na elemente koje ste prethodno proverili. Lista pri vrhu prozora sadrži spisak izraza za aktuelni Debug Inspector.

Pretraživanje modula i procesa

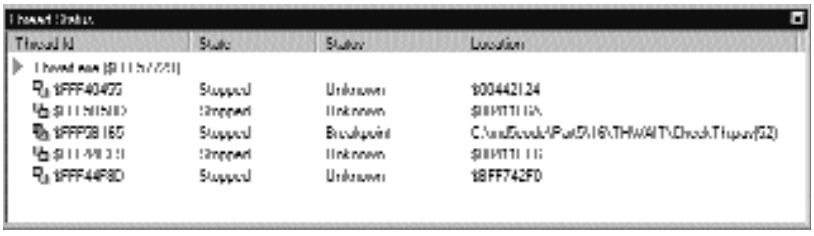
Jedna važna oblast debagera se odnosi na pretraživanje sveukupne strukture aplikacije. Prozor Modules (videti sliku 18.12) prikazuje sve izvršne module za aktuelnu aplikaciju (obično glavni izvršni fajl kao i DLL-ove koje koristi). Desni panel prikazuje spisak ulaznih tačaka različitih procedura ili funkcija svakog modula. Donji panel prikazuje spisak Pascal jedinica koje modul sadrži, ukoliko je takva informacija poznata.



SLIKA 18.12 *Prozor Modules*

Prozor Modules se može koristiti za proveru sistemskih DLL-ova i Delphi paketa samo za vreme izvršavanja koji su neophodni programu, i omogućava Vam da istražite kako se odnose prema treminima izvezenih i uvezenih funkcija. Dok alati kao što su TDump.exe ili Executable QuickView, koji su uključeni u Windows, mogu izvršiti statičku analizu EXE fajla da bi se utvrdilo koji DLL je potreban za izvršavanje programa, prozor Modules prikazuje trenutne biblioteke koje se koriste, čak i ako je neka od biblioteka dinamički učitana (videti Poglavlje 14 radi primera dinamički učitanih biblioteka). Zapamtite da možete upotrebiti prozor Modules za određivanje tačke prekida učitavanja modula, dakle, tačke prekida koja će se desiti kada sistem učita modul.

Još jedan povezan prozor je prozor Thread Status, koji prikazuje detalje svakog procesa programa koji ima više procesa. Na slici 18.3 je dat primer ovog prozora. U ovom prozoru možete promeniti aktivni proces, a možete i da operišete sa glavnim procesom. Primetićete da je ova mogućnost naročito interesantna kada debagujete dva procesa istovremeno, što je karakteristika koju možete koristiti samo na platformi Windows NT.



SLIKA 18.13 *Prozor Thread Status*

Dnevnik događaja

Još jedan zgodan debager prozor, koji je prvi put predstavljen u Delphiju 4, jeste prozor Event Log (dnevnik događaja), koji Vam omogućava da nadgledate brojne sistemske događaje: učitavanje modula, tačke prekida i njihove poruke dnevnika, Windows poruke i korisničke poruke

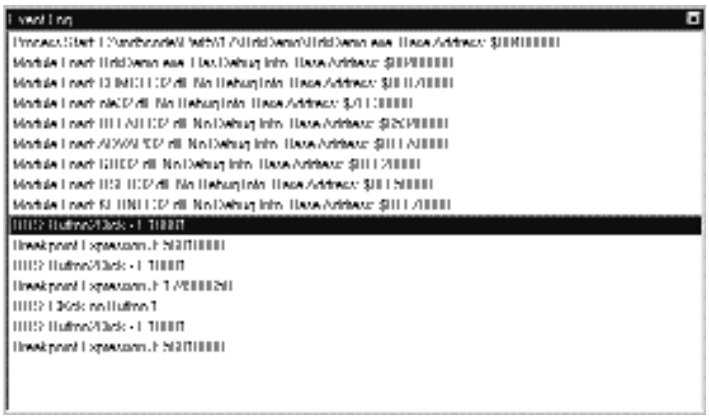
koje šalje aplikacija. Praćenje toka programa upotrebom dnevnika može biti od neprocenjive vrednosti. Kao primer, razmislite o debagovanju aplikacije kada je tajming toliko važan da se na njega može uticati ukoliko se program zaustavi u debageru. Da biste izbegli zaustavljanje programa, možete u dnevnik uneti informaciju o debagovanju da biste kasnije mogli da je pogledate.

Da biste generisali direktan dnevnik, možete ugnezditi pozive Windows API procedure `OutputDebugString` u Vaš program. Ova procedura prihvata pokazivač stringa i šalje taj string validnom uređaju za debagovanje. Prozor Event Log će presresti i prikazati tekst koji prosleđujete proceduri `OutputDebugString`. Na slici 18.14 prikazan je primer sesije debagovanja načinjene prozorom Event Log. (Usput, pozivi procedure `OutputDebugString` se pojavljuju u dnevniku kao tekst sa ODS prefiksom.) Na istoj slici takođe možete videti efekat nekih tačaka prekida koje u dnevnik unose vrednost promenljive `J`. Ovaj izlaz je preuzet iz primera OldDemo i generisan je sledećim kodom:

```
OutputDebugString (
       PChar ('Button2Click - I = ' + ItToStr (I)));
```

SAVET

Mada efekti pozivanja procedure `OutputDebugString` i zapisivanja poruke u dnevnik kao posledice tačke prekida mogu izgledati slično, ipak postoji velika razlika. Tačke prekida su spoljašnji svet za program (one su deo podrške debagera), dok se direktni stringovi moraju dodati izvornom kodu programa, potencijalno ga menjajući i dovodeći do bagova. Takođe, možda želite da ovaj kod izostavite iz konačne izrade, mada za to možete upotrebiti uslovno kompajliranje, kao što ću kasnije opisati u odeljku "Upotreba uslovnog kompajliranja za debagovane verzije i za verzije koje se prosleđuju". ■

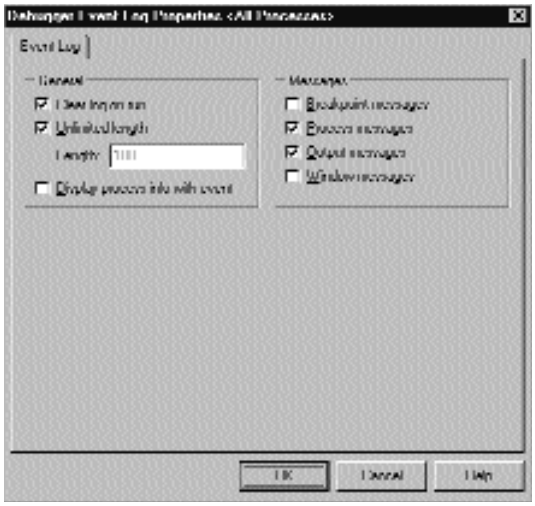


SLIKA 18.14 *Izlaz poziva procedure debagovanja OutputDebugString i informacije koje su zapisane u dnevnik tačkom prekida prozora Event Log*

Da bih dobio prikaz sa slike 18.14, ja sam onemogućio unapred određeno zapisivanje tačaka prekida u dnevnik, koje naznačava kada je program zaustavljen ili ponovo pokrenut iz tačke prekida. (Kao što je ranije istaknuto, takođe možete da naznačite zapisivanje u dnevnik kao jednu od akcija tačke prekida; onemogućavanje unapred određenog zapisivanja tačke prekida u dnevnik ne utiče na ovo zapisivanje u dnevnik.) Ja sam takođe onemogućio informacije procesa (novu

Delphi 5 opciju "Display Process Info with Event"), koja nije naročito korisna kada se debaguje jedan proces.

Event Log možete konfigurisati ovom i drugim opcijama okvira za dijalog Debugger Event Log Properties koji je prikazan na slici 18.15. Pored zapisivanja teksta poziva procedure `OutputDebugString`, podataka tačke prekida i informacija o procesu u dnevnik, prozor Event Log takođe može da presretne i sve Windows poruke koje stižu do aplikacije. Ovo je, dakle, alternativa programu WinSight koji će biti opisan kasnije u ovom poglavlju.



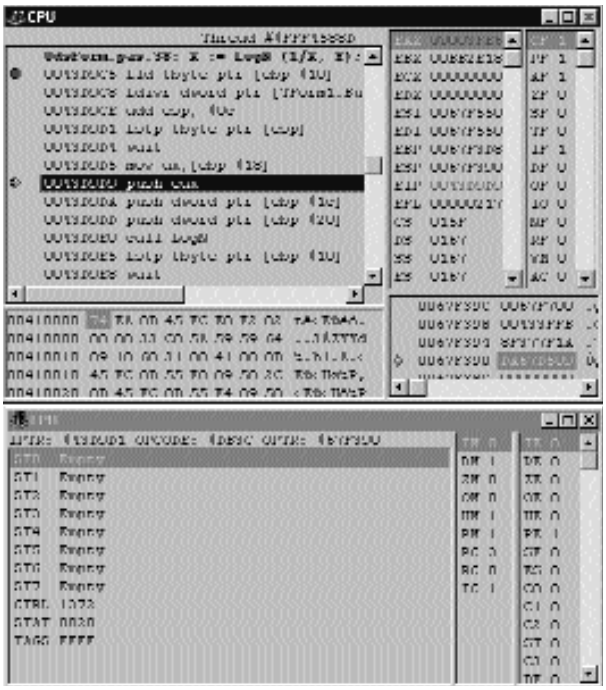
SLIKA 18.15 *Upotrebom okvira za dijalog Debugger Event Log Properties određujete koje događaje želite da pratite.*

Pravo u srž: CPU i FPU pogledi

Postoje još dva debager prozora koja nisu namenjena slabićima. Pogled CPU i novi Delphi 5 pogled FPU Vam prikazuju šta se dešava unutar centralne jedinice za obradu (Central Processing Unit — CPU, dakle, procesor) i jedinice za računanje u pokretnom zarezu (Floating Point Unit — FPU) kompjutera.

Upotreba pogleda CPU tokom debagovanja Vam omogućava da vidite dosta sistemskih informacija: vrednosti CPU registara, uključujući i specijalne zastavice i disasembler programa (naročito kada je Pascal kod uključen u komentare). Slično, pogled FPU prikazuje još registara i informacije o statusu koje se odnose na pokretni zarez i MMX podršku novijih čipova klase Pentium.

Ukoliko imate osnovno znanje o asembleru, možete upotrebiti ove informacije da biste potpuno razumeli kako Delphi kompajler prevodi Vaš izvorni kod u izvršni kod, i da biste videli efekat Delphi optimizacije kompajlera na konačni kod. Mada u početku izgleda pusto, prozor CPU programerima obezbeđuje veliku moć. Kada kliknete desnim tasterom miša i upotrebite kontekst menije, možete čak direktno promeniti vrednost CPU registara!



SLIKA 18.16 *Primer CPU i FPU pogleda*

Ostale tehnike debagovanja

Jedna od uobičajenih upotreba tačaka prekida je da se sazna kada je program stigao do određenog stanja, ali postoje i drugi načini da bi se dobila ovakva informacija. Uobičajena tehnika je da prikažete jednostavnu poruku (upotrebom procedure `ShowMessage`) na određenim linijama, samo za potrebe debagovanja. Postoje i druge ručne tehnike, kao što je promena teksta oznake na specijalnom formularu, zapisivanje u fajl dnevnika ili dodavanje linije listi ili memo polju.

Sve ove alternativne mogućnosti služe jednom od dva osnovna cilja: bilo da Vam se stavi do znanja da je određena linija koda izvršena, bilo da Vam omoguće da pogledate neke vrednosti, a u oba slučaja Vi zapravo ne zaustavljate program. Uslovno kompajliranje, alegacije i nadgledanje toka poruka su samo neke od tehnika koje možete upotrebiti da biste zamenili mogućnosti koje nudi debager.

Upotreba uslovnog kompajliranja za debagovanje i verzije koje prosleđujete

Dodavanje koda za debagovanje aplikaciji je svakako interesantno, kao što to pokazuje primer OdsDemo, ali ovaj pristup ima ozbiljan nedostatak. U konačnoj verziji programa, onoj koju dajete svojim kupcima, morate da onemogućite izlaz debagovanja, i možda je potrebno da uklonite sav kod debagovanja da biste smanjili veličinu programa i ubrzali ga. Ukoliko ste C/C++ programer, možda imate neke ideje kako da automatski uklonite programski kod. Rešenje ovog

problema je u tipičnoj C tehnici poznatoj kao *uslovno kompajliranje*. Ideja je jednostavna: napisaćete neke linije koda koje želite da kompajlirate samo u određenim situacijama, a preskočićete njihovo kompajliranje u svim drugim situacijama.

U Delphiju možete upotrebiti neke uslovne direktive kompajlera: `$IFDEF`, `$IFNDEF`, `$IFOPT`, `$ELSE` i `$ENDIF`. Na primer, u obradi događaja `Button2Click` primera OdsDemo pronaći ćete sledeći kod:

```
{$IFDEF DEBUG}
  OutputDebugString (
    PChar ('Button2Click - I =' + IntToStr (I)));
{$ENDIF}
```

Ovaj kod je uključen u kompajliranje samo ukoliko postoji simbol `DEBUG` definisan ispred linije ili ukoliko je simbol `DEBUG` definisan u okviru za dijalog Project Options. Kasnije možete da uklonite definiciju simbola, odaberete komandu Build All iz Delphijevog menija Compile i ponovo ga pokrenete. Veličina izvršnog fajla će se verovatno malo promeniti između dve verzije jer je deo izvornog koda uklonjen. Primetićete da će svaki put kada promenite definicije simbola u okviru za dijalog Project Options, biti potrebno da ponovo izradite ceo program. Ukoliko ga samo pokrenete, starija verzija će biti izvršena jer će izvršni fajl izgledati kao da je ažuriran u poređenju sa izvornim fajlovima.

UPOZORENJE

Uslovno kompajliranje za svrhe debagovanja treba da koristite veoma pažljivo. Zapravo, ukoliko debagujete program ovom tehnikom i kasnije izmenite kod (prilikom uklanjanja `DEBUG` definicija), možete načiniti nove bagove ili možete pronaći bagove koji su bili sakriveni procesom debagovanja. Zbog toga je, u opštem slučaju, bolje da pažljivo debagujete konačnu verziju Vaše aplikacije, i da ne načinite više nikakve izmene u kodu. Prevelika upotreba direktive `IFDEF` takođe uništava održavanje koda i čitljivost. ■

Upotreba alegacija

Alegacije (assertations) su tehnika koju u Delphiju možete upotrebiti za debagovanje. Alegacija je u osnovi izraz koji uvek treba da bude tačan, jer je deo logike programa. Na primer, mogao sam da osiguram da broj korisnika mog programa uvek bude bar jedan jer se moj program ne može izvršavati bez korisnika. Kada je alegacija netačna, to znači da postoji nedostatak u kodu programa (u *kodu*, ne u izvršavanju).

Jedini parametar procedure `Assert` je Boolean uslov koji želite da testirate. Ukoliko je uslov ispunjen, program može da nastavi izvršavanje kao i obično; ukoliko uslov nije ispunjen (ukoliko alegacija ne uspe), program se poziva na izuzetak `EAssertionFailed`. Evo primera iz programa Assert:

```
procedure TForm1.BtnIncClick (Sender: TObject);
begin
  if Number < 100 then
    Inc (Number);
  ProgressBar1.Position := Number;
  // test the condition
  Assert ((Number > 0) and (Number <= 100));
end;
```

Druga kontrola formulara Assert ovog primera generiše kod koji je delimično netačan, tako da alegacija može da ne uspe, a efekat je prikazan na slici 18.17. Imajte na umu da su alegacije alat za debagovanje. Trebalo bi ih koristiti da biste se uverili da je kod programa korektan. Korisnici nikada ne bi trebalo da vide da alegacija ne uspeva, bez obzira na to šta se dešava u programu i bez obzira na to koje podatke unose, jer ukoliko alegacija ne uspe, to znači da verovatno postoji greška u Vašem kodu. Da biste testirali specijalne uslove greške, potrebno je da koristite izuzetke, ne alegacije.

SLIKA 18.17 *Poruka o grešci alegacije (iz primera Assert)*

Alegacije su toliko blisko povezane sa debagovanjem i testiranjem da ćete u opštem slučaju želeti da ih uklonite upotrebom direktive kompajlera `$ASSERTIONS` ili `$C`. Jednostavno dodajte liniju `{$C-}` negde u fajlu izvornog koda, i ta jedinica će biti kompajlirana bez alegacija. Ovim se ne onemogućavaju alegacije, već se se zapravo uklanja odgovarajući kod iz programa. Alegacije, takođe, možete isključiti sa strane Compiler okvira za dijalog Project Options.

UPOZORENJE

Nemojte zaboraviti da ponovo izradite svoj projekat posle izmene podešavanja alegacija, ili podešavanja neće imati efekta. ■

Pregled toka poruka

Integrisani debager obezbeđuje uobičajeni način za pregled izvornog koda programa. U Windowsu, ipak, ovo često nije dovoljno. Kada želite da razumete detalje o interakciji između Vašeg programa i okruženja, često će Vam biti potreban alat za praćenje poruka koje sistem šalje aplikaciji. Ovo možete obaviti u integrisanom debageru upotrebom prozora Event Log i aktivirajući Windows poruke u okviru za dijalog Event Log Properties. Primetićete da ovakav tip zapisivanja u dnevnik nije unapred određen.

Event Log ipak nije dovoljno fleksibilan jer ne možete odabrati kategorije poruka ili odredišne prozore: dobićete kompletan dnevnik svih poruka prozora Vašeg programa, što je često ogromna lista. Alternativni alat za praćenje Windows poruka je WinSight, višenamenski alat koji je uključen u Delphi. Drugi slični alati su Vam na raspolaganju iz raznih izvora, uključujući knjige i članke iz časopisa. Ja sam izradio spostvenu verziju koju ćete malo kasnije videti.

Da biste postali ekspert za Delphi programiranje, morate naučiti da prostudirate tok poruka koji sledi ulaznu akciju korisnika. Kao što znate, Delphi programi (kao i uopšte Windows aplikacije) su vođeni događajima. Kod se izvršava kao odgovor na događaj. Windows poruke su ključni element koji stoji iza Delphi događaja, mada ne postoji korespondencija jedan-na-jedan između ovih događaja. U Windowsu postoji više poruka nego u Delphiju, ali neki Delphi događaji se odigravaju na višem nivou nego Windows poruke. Na primer, Windows obezbeđuje samo

ograničenu podršku za prevlačenje mišem, dok Delphi komponente nude potpuni skup događaja za prevlačenje mišem.

Upotreba WinSighta

WinSight je Borlandov alat koji Vam je na rapolaganju u Delphi/Bin direktorijumu. Može se koristiti za izradu hijerarhijskog grafa postojećih prozora i za prikazivanje detaljnih informacija o toku poruke. Naravno, WinSight ne zna ništa o Delphi događajima, te ćete morati da sami odgonetnete korespondenciju između mnogih događaja i poruka (ili ćete morati da prostudirate VCL izvorni kod ukoliko ga imate). WinSight Vam može prikazati, u čitljivom formatu, sve Windows poruke koje dolaze do prozora, naznačavajući odredišni prozor, njegov naslov i klasu i njegove parametre. Vi možete koristiti komandu Options iz WinSightovog menija Message da biste izdvojili neke od poruka i da biste prikazali samo grupe za koje ste zainteresovani.

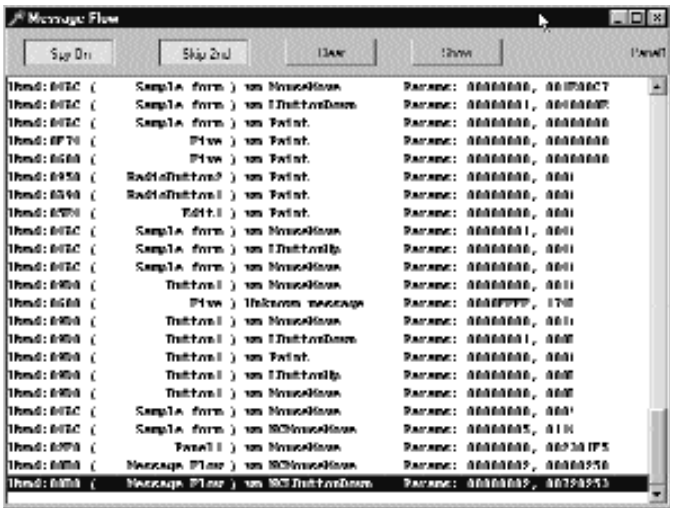
Obično, za Delphi programere nadgledanje toka poruka može biti korisno kada su suočeni sa nekim bagovima koji se odnose na redosled aktiviranja i deaktiviranja prozora ili na primanje ili gubljenje ulaznog fokusa (događaji `OnEnter` i `OnExit`), naročito kada su uključeni prozori poruka ili drugi prioritetni prozori. Ovo je uobičajena problematična oblast i često, pregledajući tok poruka, možete videti šta je pošlo naopako. Takođe, možete poželeti da pregledate tok poruka kada direktno obrađujete Windows poruke (umesto da upotrebite obrade događaja). Upotrebom programa WinSight možete dobiti više informacija o tome kada poruka stiže i kakve parametre nosi.

Pogled na poslate poruke

Drugi način za pregled toka poruka je da direktno "ulovite" neke Windows poruke u Delphi aplikacijama. Ukoliko ovu analizu ograničite na poslate poruke (koje se šalju metodom `PostMessage`) i zanemarite upućene poruke (koje se šalju metodom `SendMessage`), analiza postaje gotovo trivijalna jer možemo upotrebiti događaj `OnMessage` klase `TApplication` i komponente `TApplicationEvents`. Ovaj događaj je namenjen tome da aplikaciji da mogućnost da izdvoji poruke koje dobija i da određene poruke obradi na specijalan način. na primer, možete događaj koristiti za obradu poruka prozora koji je povezan sa samim objektom `Application`, koji nema specifične obrade događaja, kao što smo to učinili u primeru SysMenu2 u Poglavlju 6.

U primeru MsgFlow pogledaćemo sve poruke koje su izdvojene iz reda poruka aplikacije (to jest, poslate poruke). Opis svake poruke se dodaje u listu koja se nalazi na formularu primera. Za ovu listu sam upotrebio font Courier jer nije proporcionalan, ili jednako razmaknut (monospaced) font, tako da se izlaz formatira poljima koja su pravilno poravnata u listi. Kontrole na paleti alata se mogu upotrebiti za uključivanje ili isključivanje prikazivanja poruka, za uklanjanje poruka iz liste i za zanemarivanje poruka koje se uzastopno ponavljaju. Na primer, ukoliko pomerite pokazivač miša, dobićete mnogo uzastopnih poruka `wm_MouseMove`, koje se mogu preskočiti a da ne izgubite mnogo informacija.

Da bi Vam se omogućilo da načinite nekoliko pravih testova, program sadrži drugi formular (koji se prikazuje kada kliknete četvrtu kontrolu), na kojem se nalaze razne komponente (koje su nasumično odabrane). Ovaj formular možete upotrebiti da biste videli tok poruka standardnog Delphi prozora. Na slici 18.18 prikazan je primer izlaza programa MsgFlow kada je drugi formular vidljiv. Izvorni kod programa (koji je prilično dugačak) se ne opisuje u tekstu, ali Vam je na raspolaganju uz ostale primere ove knjige.



SLIKA 18.18 *Program MsgFlow u vreme izvršavanja i kopija drugog formulara*

Memorijski problemi

Jedan od najvećih problema kada debagujete Delphi program je provera šta se dešava sa memorijom aplikacije i memorijom sistema. Dva od najčešćih memorijskih problema su nedostatak (leak) memorije (ne oslobađnje neupotrebljene memorije, tako da će program koristiti mnogo više memorije nego što mu je zapravo potrebno) i prevelika zauzetost memorije (`overrun` upotreba memorije koja se već koristi i referenciranje na objekat koji je već uklonjen).

Postoji više pristupa u Delphiju koje možete upotrebiti za detektovanje i rešavanje ovih memorijskih problema, ali nema mnogo pomoći koju možete dobiti od integrisanog debagera. Da biste detektovali ove probleme, možete upotrebiti tehnike koje su opisane kasnije u ovom odeljku, ili neke alate nezavisnih programera koji se opisuju na kraju ovog poglavlja. Ono na šta želim da obratim pažnju je pregled različitih memorijskih oblasti, tako da možete bolje razumeti kada će ovi memorijki problemi izaći na površinu i kako da načinite preventivni pristup da biste ih sasvim izbegli.

Procesi i memorija

Nije lako detaljno analizirati upravljanje memorijom za Delphi aplikacije, jer treba uzeti u obzir mnogo informacija. Prvo, postoji Windowsovo upravljanje memorijom, koje je na paltformi Win32 prilično jednostavno i robusno za aplikacije, ali mnogo složenije za DLL-ove. Na nivou aplikacije postoji Delphijevo upravljanje memorijom.

U palatformi Win32 svaka aplikacija vidi svoju lokalnu memoriju kao jedan veliki segment od 4GB, bez obzira na količinu fizičke memorije koja je na raspolaganju. Ovo je moguće jer operativni sistem mapira virtuelne memorijske adrese za svaku aplikaciju u fizičke RAM adrese i po potrebi prebacuje ove blokove memorije na disk (automatski učitavajući odgovarajuću stranu iz

swap fajla u memoriju). Ovim jednim ogromnim memorijskim segmentom upravlja operativni sistem u segmentima od po 4KB koji se nazivaju strane (pages).

NAPOMENA

Svaki proces sadrži svoj privatni adresni prostor, koji je u potpunosti odvojen od drugih adresnih prostora. Ovo operativni sistem čini robusnijum nego što je to bio u danima 16-bitnih Windowsa, kada su sve aplikacije delile isti adresni prostor. Nedostatak je to što je sada mnogo teže razmenjivati informacije između aplikacija. ■

Zapravo, i u Windowsu 95/98 i u Windowsu NT aplikacija može direktno upravljati samo polovinom svog adresnog prostora (2GB), dok je druga polovina rezervisana za operativni sistem. Na sreću, 2GB je obično više nego dovoljno.

Drugi važan element Win32 upravljanja memorijom je alokacija virtuelne memorije. Pored alociranja memorije, proces jednostavno može rezervisati memoriju za buduću upotrebu (upotrebom operacija niskog nivoa koje se nazivaju virtuelne alokacije). Na primer, u Delphi aplikaciji možete upotrebiti proceduru `SetLenght` za rezervisanje prostora za string. Delphi čini istu stvar neprimetno kada kreirate ogroman niz. Ova memorija neće biti alocirana — samo će biti rezervisana za buduću upotrebu. U praksi to znači da memorijski podsistem neće koristiti adrese iz tog opsega za druga alociranja memorije.

Na sreću, veći deo upravljanja memorijom, kako na nivou aplikacije tako i na nivou sistema, potpuno je neprimetan za programere. Zbog toga Vi obično nemate potrebu da znate detalje o funkcionisanju strana memorije, i ovu temu nećemo dalje razmatrati u ovoj knjizi. Umesto toga, pozabavićemo se statusom dela memorije, nečim što ćete videti da je korisno kada pišete i debagujete aplikaciju.

Globalni podaci, stek i kolekcija (heap)

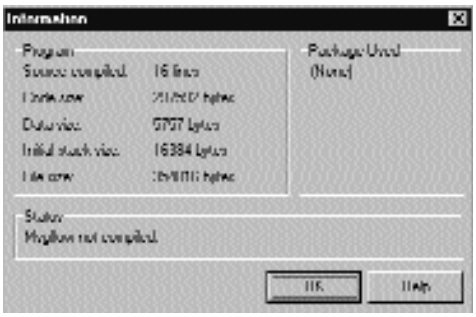
Memorija koju koristi određena Delphi aplikacija se može podeliti u dve oblasti: kod i podatke. Delovi izvršnog fajla programa, njegovih resursa (bitmape i DFM fajlovi) i biblioteka koje koristi program, učitavaju se u memorijski prostor programa. Ovi memorijski blokovi su samo za čitanje i mogu se deliti između više procesa.

Mnogo je interesantnije pregledati deo za podatke. Podaci Delphi programa se čuvaju u tri jasno razdvojene oblasti: globalnoj memoriji, steku i kolekciji (heap).

Globalna memorija

Kada Delphi kompajler generiše izvršni fajl, on određuje prostor koji je potreban za čuvanje promenljivih koje postoje tokom celog života programa. Globalne promenljive koje su deklarisane u interfejs odeljku ili implementacionom odeljku jedinice spadaju u ovu kategoriju. Primetićete da, ukoliko je globalna promenljiva tipa klase, u globalnoj memoriji se čuva samo 4-bajtna referenca objekta.

Veličinu globalne memorije možete odrediti upotrebom elementa menija Project➥Information posle kompajliranja programa i pregledanjem vrednosti koja se odnosi na veličinu podataka. Na slici 18.19 je prikazana upotreba gotovo 6K globalnih podataka, što nije mnogo kada se uzme u obzir da uključuje globalne podatke VCL-a i Vašeg programa.



SLIKA 18.19 *Informacije o kompajliranom programu koje prikazuje Delphi*

Stek

Stek (stack) je dinamička memorijska oblast, koja se alocira i dealocira prema LIFO redosledu: Last In, First Out (poslednji unutra, prvi napolje). To znači da će poslednji memorijski objekat koji ste alocirali biti prvi objekat koji će biti uklonjen. Stek memoriju tipično koriste rutine (pozivi procedura, funkcija i metoda) za prosleđivanje parametara i njihovih povratnih vrednosti i za promenljive koje Vi deklarišete u okviru rutine. Kada se završi poziv rutine, oslobađa se njena memorija na steku. Zapamtite da se upotrebom unapred određene Delphi konvencije pozivanja registara, parametri prosleđuju CPU registrima umesto da se prosleđuju steku.

Windows aplikacije mogu rezervisati veliku količinu memorije za stek. U Delphiju Vi određujete ovaj parametar na strani Liker okvira za dijalog Project Options. Ipak, unapred određena vrednost je u opštem slučaju dovoljna. Ukoliko dobijete poruku da je stek pun, to je verovatno zbog toga što imate funkciju koja beskonačno rekurzivno poziva samu sebe, a ne zato što je prostor steka premali. Inicijalna veličina steka je još jedna informacija koju možete dobiti upotrebom elementa menija Project➥Information.

Kolekcija (heap)

Kolekcija (gomila — heap) je oblast u kojoj se alociranje i dealociranje memorije nasumično obavlja. To znači da ukoliko alocirate tri bloka memorije zaredom, kasnije se mogu ukloniti bilo kojim redom. Menadžer kolekcije se stara o svim detaljima, te možete jednostavno zatražiti novu memoriju upotrebom `GetMem` ili pozivanjem konstruktora za kreiranje objekta, a Delphi će Vam dati novi memorijski blok (verovatno ponovo koristeći memorijske blokove koji su već oslobođeni). Delphi koristi kolekciju za alociranje memorije za svaki objekat, tekst stringova, dinamičke nizove i za ostale specifične zahteve za dinamičkom memorijom.

Kako je priroda kolekcije dinamička, kolekcija je memorijska oblast u kojoj programi najčešće imaju probleme. Delphi koristi brojne tehnike za rukovanje memorijom, uključujući prebrojavanje referenci (za stringove, dinamičke nizove ili promenljive objekta tipa interfejsa) i vlasništvo (za VCL komponente). Razumevanje ovih tehnika i njihovo pravilno primenjivanje su osnova za pravilno upravljanje dinamičkom memorijom. Da biste proverili da li sve pravilno funkcioniše i da biste shvatili šta je pošlo naopako, Delphijev debager Vam neće mnogo pomoći. I Windows i Delphi prikazuju status memorije (iz različitih perspektiva, doduše), što Vam omogućava da proučite trenutnu situaciju. Delphi takođe prikazuje svoje interno upravljanje memorijom, tako

da ga možete proučiti ili ga potpuno zameniti. Možete čak promeniti rukovanje memorijom za određene klase, zaobilaženjem metoda klase koji su zaduženi za alociranje i dealociranje memorije.

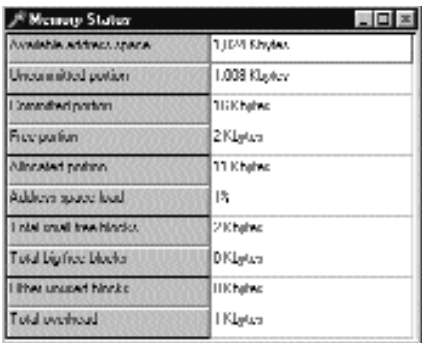
Praćenje memorije

Windows API sadrži nekoliko funkcija koje nam omogućavaju da proverimo status memorije. Najmoćnije od ovih funkcija su deo takozvanog ToolHelp API-ja (nema nikakve veze za Delphijevim ToolsAPI-jem) i zavise od platforme: postoje ili u Windowsu 98 ili u Windowsu NT — ne u oba slučaja.

Ukoliko želimo da ostanemo na poznatom terenu, možemo koristiti `GlobalMemoryStatus`, funkciju koja nam omogućava da proverimo status memorije celokupnog operativnog sistema. Ova funkcija kao rezultat daje sistemske informacije o fizičkom RAM-u, strani fajla (ili swap fajla) i globalnom adresnom prostoru. Da bih demonstrirao upotrebu ove funkcije, ja sam izradio program MemIcon, koji je opisan u Poglavlju 19, jer je njegov ključni element prikazivanje upotrebe ikona u polju koje se nalazi na TaskBaru (liniji aktivnih programa).

U opštem slučaju, informacije koje se odnose na status Windows memorije malo interesuju Delphi programere, naročito ukoliko ih poredite sa detaljnim informacijama o Delphijevom upravljanju memorijom koje daje VCL funkcija `GetHeapStatus`. Ova funkcija je definisana u jedinici System (ili jedinici ShareMem) i kao rezultat daje strukturu sa priličnom količinom informacija: adresni prostor koji je virtuelno alociran; prostor koji je rezervisan ili je rezervisanje ukinuto, fizički alociran prostor, slobodan (razdvajajući velike i male blokove memorije), ili nije u upotrebi; i ukupna memorija koja se koristi za upravljanje memorijom.

Podatke koje daje ova funkcija možete videti u prozoru MemoryStatus koji primer VclMem prikazuje kao svoj okvir za dijalog About. Ovaj prozor je prikazan na slici 18.20. Njegov formular sadrži komponentu sa tabelom stringova, koja se automatski ažurira kada se program pokrene i upotrebom tajmera (tako da ovaj prozor možete ostaviti otvoren i prikazivati informacije koje se periodično ažuriraju). Pored informacija o memoriji koje se prikazuju, program nije naročito interesantan; ovaj formular bi trebalo da dodate nekoj od Vaših složenih aplikacija, tako da možete testirati status memorije.



SLIKA 18.20 *Prozor Memory Status jednostavnog primera VclMem. Ovaj formular bi trebalo da dodate Vašim programima da biste proverili koliko memorije koriste*

NAPOMENA

Ovaj program je redukovana verzija test primera koji možete naći u knjizi "Delphi Developer's Handbook" (Sybex, 1998). U toj knjizi objašnjenja su mnogo detaljnija i razmatraju se ne samo slučajevi kada upravljanje memorijom može dovesti do problema, već i kako da napišete svoje upravljanje memorijom. Možda želite da to učinite da biste zamenili Delphijevu šemu upravljanja memorijom ili da biste ste se povezali sa upravljanjem memorijom: na primer, da biste prebrojali memorijske blokove koji su alocirani ili dealocirani ili da proverite nedostatke memorije. ■

Alati nezavisnih programera

Delphijev integrisani debager, samostalni Turbo Debugger i udaljeni debager su velika pomoć prilikom pronalaženja grešaka u izvornom kodu, ali Vam neće mnogo pomoći kada su u pitanju problemi sa memorijom, a i još su prilično ograničeni u nekim oblastima. Pored tehnika koje sam ovde razmatrao, postoji i nekoliko alata nezavisnih programera koji mogu biti od izuzetne pomoći u pronalaženju grešaka i rešavanju memorijskih problema. U ovom odeljku ću navesti nekoliko alata i ukratko ću istaći njihove karakteristike, da bih Vam dao ideju šta je na raspolaganju.

Memory Sleuth

Memory Sleuth je proizvod Turbo Power Software Company, Inc. (`http://www.turbopower.com`). Prvobitno je razvijen za Delphi 1, programer je Per Larsen (Per Larsen), i dat je kao MemMonD32. Posle kompajliranja Vašeg Delphi programa, jednostavno ga izvršite kroz Memory Sleuth (učitavajući ga i izvršavajući iz ovog okruženja umesto iz Delphi IDE-a).

Alat detektuje memoriju i Windows nedostatke resursa, dajući detaljan prikaz linija izvornog koda koje prouzrokuju problem. Ovaj alat se povezuje sa Delphijevim upravljanjem memorijom i nadgleda alociranje Delphi objekata, ali takođe proverava status Windows memorije. Pored generisanja izveštaja o problemima, program takođe može da detektuje vrhunac upotrebe memorije i resursa, pa čak i da nacrta neke lepe grafikone. Poslednja poboljšana verzija dodaje još neke mogućnosti ovom alatu.

CodeSite

CodeSite je proizvod Raize Software Solutins, Inc. (`http://www.raize.com`). Autor je Rej Konopka (Ray Konopka) (koji takođe programira Raize Components). Po rečima autora, "CodeSite je napredni Delphi alat za debagovanje, zasnovan na starom pristupu slanja poruka od aplikacije do programa za prikazivanje poruka. Ipak, za razliku od svojih prethodnika, CodeSite obrađuje mnogo više stvari od prostih poruka."

Zapravo, možete poslati svojstva i cele objekte u prozor za debagovanje, koji celu operaciju čini veoma brzom, a da ne utiče na kod programa. Umesto upotrebe standardnog prozora za debagovanje potrebno je da koristite prozor koji obezbeđuje CodeSite, u kojem se prikazuju detaljne informacije, i koji Vam omogućava, na primer, da uporedite dva prikaza istog objekta koja su načinjena u različtom trenutku.

BoundsChecker

BoundsChecker je dobro poznati Windows alat za detekciju grešaka koji proizvodi NuMega Technologies, Inc. (`http://www.numega.com`). Program ima dugu tradiciju kod Microsoft C++ programera, a već dugo je na raspolaganju za Delphi. BoundsChecker nadgleda sve Windows API pozive koje čini Vaš program ili VCL, prati pogrešne parametre, nedostatke resursa, stek memoriju i memoriju kolekcije (heap), i još mnogo toga. Alat se može koristiti za najnoviji Windows API uključujući Win32, ActiveX, DirectX, COM, Winsock i Internet API.

Jednostavno pokrenite program BoundsChecker, učitajte svoj izvršni fajl (koji mora biti kompajliran sa informacijama za debagovanje i okvirima za stek) i pokrenite ga. Svaki put kada se detektuje greška, svi detalji se zapisuju u dnevnik, tako da na kraju sesije debagovanja možete pronaći detalje. BoundsChecker možete koristiti i za testiranje kompatibilnosti Vašeg programa sa raznim verzijama Win32 API-ja, ukoliko mislite da će se javiti problemi u Windowsu 98 ili Windowsu NT.

Šta je sledeće?

U ovom poglavlju ste videli da postoje brojni alati koje možete koristiti za debagovanje Delphi aplikacija, kako integrisanih tako i onih koje se odnose na Windows sistem. Windows aplikacije ne žive samostalno. Čvrsto su povezane sa sistemom i, obično manje direktno, sa ostalim aplikacijama koje se izvršavaju. Prisustvo drugih Windows aplikacija može uticati na performanse Vaših programa kao i na njihovu stabilnost.

U narednom poglavlju ću razmatrati brojne tehnike koje se odnose na štampanje, upotrebu resursa, manipulisanje fajlovima, pristupanje Clipboardu, upotrebu Windows ini fajlova, upotrebu Registryja, kreiranje i povezivanje help fajlova i kreiranje instalacionog programa, kao i nove Delphi 5 karakteristike kao što su TeamSource i Integrated Translation Environment. Svaka tehnika predstavlja koristan pristup rešavanju specifičnih programskih problema.

POGLAVLJE

19

Još Delphi tehnika

U prethodnim poglavljima smo razmatrali glavne karakteristike Delphija. U ovom poglavlju ćemo našu pažnju usmeriti na niz praktičnih zadataka koje morate uzeti u obzir u svakodnevnom radu. Proučićemo odgovarajuće tehnike Delphija kojima se implementira svaki od ovih zadataka, i demonstrirаćemo ih primerima.

Pošto postoji mnogo tema koje treba proučiti (a odnose se na ovu oblast), u ovom poglavlju ćemo ih kratko opisati, a primere ću predstaviti uz minimum objašnjavanja. Kao i obično, možete detaljno proučiti preuzete fajlove izvornog koda.

Upravljanje Windows resursima

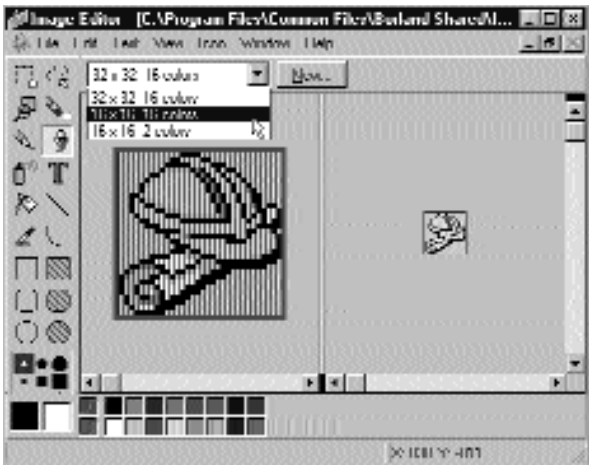
Windows resursi igraju važnu ulogu, gledano iz perspektive upotrebe memorije. Resursi se čuvaju u zasebnim blokovima u izvršnom fajlu aplikacije, tako da se ti blokovi učitavaju u memoriju po zahtevu, da se mogu odbaciti i da se resursi mogu koristiti samo kao podaci samo za čitanje.

Pre nego što se pozabavimo specijalnim upotrebama resursa u Delphiju, pogledajmo alate koje možete upotrebiti za pripremu resursa. Delphi sadrži Image Editor koji možete koristiti za manipulisanje bitmapama, ikonama i kursorima, mada ćete u nekim slučajevima više voleti da upotrebite editor resursa kakav je Borlandov Resource Workshop (koji je sada deo Delphija) ili neki drugi editor resursa.

Upotreba editora resursa

Delphijev Image Editor možete da aktivirate iz menija Tools. Image Editor Vam omogućava da manipulišete sa četiri vrste fajlova. Tri od četiri tipa fajlova su fajlovi koji sadrže specifične tipove resursa (ICO, CUR i BMP), dok je poslednji format fajla za kompajlirane resurs fajlove (RES), koji može da sadrži sva tri tipa grafičkih resursa. Pojedinačni RES fajlovi mogu da sadrže jedan ili više resursa bilo kog tipa (uključujući grafičke tipove resursa).

U Image Editoru možete pripremiti bilo koji tip ikone, kursora ili bitmape. Jedan resurs ikone može da sadrži više bitmapa različitih veličina i boja. Ikona obično ima standardnu 32x32 sliku ili 16x16 sliku (mala slika). Na slici 19.1 možete videti primer ikone sa više definisanih slika (samo je jedna prikazana) unutar Delphijevog Image Editora. Kursor, nasuprot tome, može da ima samo jednu sliku, ali morate da odredite *hot-spot* poziciju kursora, da biste odredili koja tačka slike je aktivna tačka.



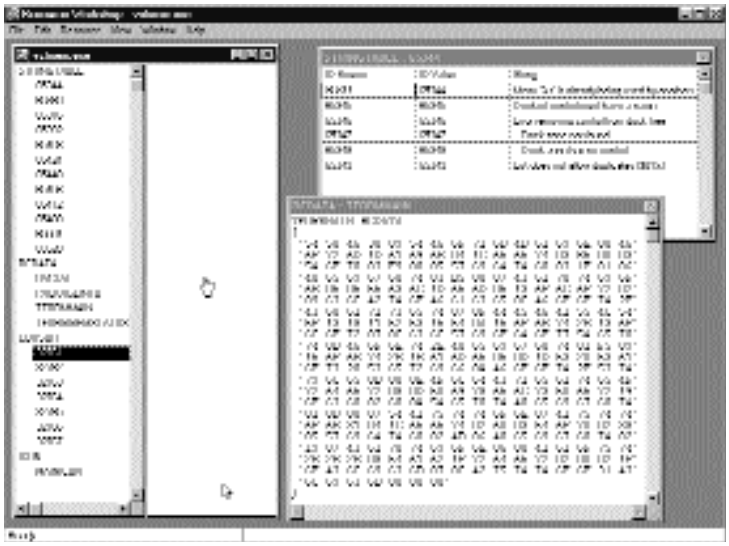
SLIKA 19.1 *Delphijev Image Editor sa različitim tipovima slika koje možete definisati za resurs ikone*

U osnovi postoje dva načina na koje možete koristiti Image Editor:

- Možete pripremiti specifične fajlove (naročito bitmape i ikone) koje će se učitavati u Delphi okruženje u vreme dizajniranja (upotrebom svojstava) ili u vreme izvršavanja (upotrebom metoda `LoadFromFile` u Vašem kodu).
- Možete pripremiti resurs fajlove koji sadrže više resursa i učitati resurse u vreme izvršavanja upotrebom Windows API poziva, što će biti opisano u narednom odeljku. Kada u Image Editoru radite sa resurs fajlovima, pogled Tree Vam omogućava da prikažete listu elemenata svake grupe.

Image Editor je koristan alat, ali su njegove mogućnosti ograničene. Kada Vam je potreban moćniji editor resursa, možete upotrebiti Borlandov Resource Workshop (koji možete pronaći na instalacionom CD-u Delphija 5). Resource Workshop Vam omogućava da otvorite bilo koji fajl sa resursima. Ovaj alat takođe možete upotrebiti da biste izdvojili resurse iz kompajliranog programa, DLL-a ili bilo kog drugog izvršnog fajla (što ne znači da je to uvek legalno; postarajte se da odredite kakva su prava na bilo koju sliku koju želite da upotrebite).

Ukoliko otvorite Delphi aplikaciju koristeći Resource Workshop, otkrićete da zapravo sadrži niz resursa, pored ikone koja se nalazi u RES fajlu. Unapred je određeno da Delphijev izvršni fajl sadrži tabelu stringova sa sistemskim porukama, naslovima i drugim generičkim stringovima (kao što su nazivi meseci), binarne podatke iz korisničkih resursa (koji opisuju formulare u formatu `RCDATA`), neke kursore i jednu ikonu. Na slici 19.2 je prikazana lista resursa za primer VclMem iz prethodnog poglavlja, a to je relativno jednostavan program. Možete videti kom pletnu listu, sa aktivnim kursorom, delom tabele stringova i binarnim podacima formulara. Složeniji programi mogu da sadrže mnogo više resursa, u zavisnosti od broja formulara, VCL jedinica koje sadrže, ikona i bitmapa koje dodajete projektu i tako dalje.



SLIKA 19.2 *Lista resursa kompajliranog Delphi programa u Resource Workshopu*

SAVET

Delphi sadrži interesantan program, nazvan Resource Explorer, koji Vam omogućava da otvorite izvršni fajl (EXE ili DLL) i da pogledate većinu njegovih resursa, baš kao i u Resource Workshopu. Resource Explorer nema integrisane editore resursa, ali uz pomoć ovog programa lako možete kopirati resurse i preneti ih u fajl resursa Vaše Delphi aplikacije, koristeći drugi editor resursa. ■

Učitavanje resursa

Najjednostavniji način za pristupanje grafičkom fajlu je njegovo učitavanje u svojstvo. Trenutno jedini resursi koje možete da učitate upotrebom svojstava su ikone i bitmape, a ponekad i metafajlovi. Na primer, komponentu Image možete smestiti kao pozadinu okvira za dijalog i u nju učitati bitmapu. U ovom slučaju bitmapa je umetnuti resurs aplikacije, što znači da nije potrebno da šaljete originalni BMP fajl (nešto što je potrebno da učinite ukoliko se bitmapa učitava u sliku u vreme izvršavanja, pozivanjem metoda slike `LoadFromFile`). Ipak, slika se ne dodaje izvršnom fajlu kao samostalni bitmapirani resurs. Bitmapa se uključuje u binarni resurs koji predstavlja formular.

Ovakav pristup otežava beskrupuloznim ljudima da koriste editor resursa da bi ukrali Vašu bitmapu. Ipak, imajte na umu da je moguće izdvojiti Delphi resurs formulara upotrebom alata kakav je Resource Workshop, sačuvati taj resurs u fajlu RES formata (ali sa ekstenzijom DFM), a zatim ga ponovo učitati kao DFM fajl u Delphi editor. Da bi se to učinilo, neko mora da zna da je Vaša aplikacija načinjena uz pomoć Delphija. Nasuprot ovome, ikone formulara i aplikacije se smeštaju u kompajlirani fajl u standardnom formatu resursa da bi se aplikacijama kakve su Explorer ili Windows 95 školjka omogućilo da ih izdvoje i koriste kao nagoveštaj za korisnika.

Druga tehnika koju možete da koristite za pristupanje resursima u Delphiju je manuelni pristup. Za ovaj metod morate prvo da definišete zaseban fajl resursa sa resursima koji su Vam potrebni. Drugi korak je da uključite fajl resursa u projekat, upotrebom direktive kompajlera `$R`. Zapravo, nasuprot tipičnom C/C++ pristupu, Delphi projekti mogu da imaju veliki broj fajlova resursa.

UPOZORENJE

Nemojte prilagođavati unapred određeni fajl resursa — fajl koji ima isti naziv kao i projekat — jer Delphi ponekad menja taj fajl, te možete izgubiti načinjena prilagođavanja. Jednostavno dodajte druge RES fajlove u trenutni direktorijum i dodajte direktivu kompajlera `$R` bilo gde u izvornom kodu da biste ih učitali. U Delphiju 5 možete dodati i RC fajlove u projekat koristeći Project Manager. Ovaj fajl će automatski biti kompajliran kao RES fajl i biće povezan u izvršni fajl programa, čak i kada se na njega ne referiše direktivom kompajlera `$R`. ■

Kada ste definisali resurse i kada ste ih uključili u svoju aplikaciju, možete upotrebiti sledeće Windows API funkcije da biste učitali resurse: `LoadAccelerators`, `LoadBitmap`, `LoadCursor`, `LoadIcon`, `Loadmenu`, `LoadResource` i `LoadString`. Svaka od ovih funkcija učitava specifični tip resursa, izuzev funkcije `LoadResource`, koja se koristi za korisničke resurse. Prvi parametar ovih funkcija je hendl instance aplikacije, koja se u Delphiju čuva u globalnoj promenljivoj `HInstance`. Drugi parametar je naziv resursa koji želite da učitate. Naravno, svaka aplikacija može imati veliki broj ikona, bitmapa i drugih resursa kojima se može pristupiti prema nazivu.

Funkcija `LoadString` sadrži i neke druge parametre kojima se naznačava bafer u koji želite da kopirate string i veličinu ovog bafera. Druge funkcije za učitavanje jednostavno kao rezultat daju

hendl na učitani resurs. Ovaj hendl možete direktno dodeliti svojstvu `Handle` odgovarajućeg VCL objekta, ili (još bolje) upotrebiti specifične metode VCL objekata kao u sledećem kodu:

```
var
  Bmp: TBitmap;
begin
  Bmp := TBitmap.Create;
  Bmp.LoadFromResourceName(HInstance, 'MyBitmap');
```

Proučite primer Mines koji se nalazi u bonus poglavlju (na adresi `www.sybex.com`), koji u potpunosti prikazuje upotrebu bitmapa u fajlovima resursa.

Ikone za aplikacije i formulare

U Delphiju svaka aplikacija i svaki formular imaju sopstveno svojstvo Icon. Kada ne odredite ovo svojstvo formulara, program jednostavno koristi vrednost svojstva `Icon` objekta `Application`. Ovu unapred određenu ikonu možete pogledati i izmeniti pomoću strane Application okvira za dijalog Project Options. Ova ikona se takođe koristi na Windowsovom Taskbaru, jer je prozor koji se prikazuje na Taskbaru za Delphi aplikaciju sakriveni prozor objekta `Application`.

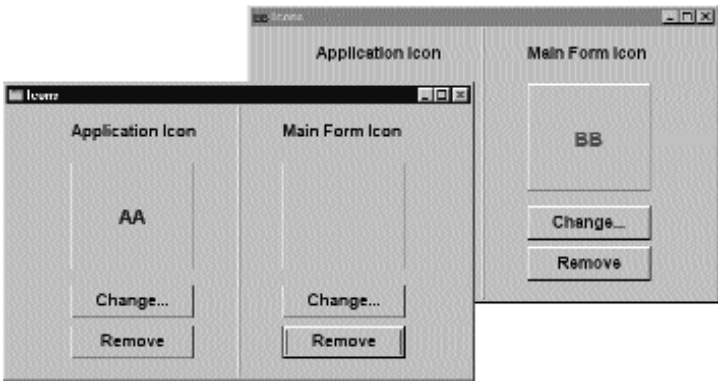
Svi ovi scenariji su prikazani primerom jednostavnog programa koji sam napisao i nazvao Icons. Formular ovog primera je podeljen na dva dela: na levoj strani je oznaka, komponenta Image i dve kontrole koje se odnose na ikonu aplikacije. Takođe, postoji i komponenta OpenDialog koju ćemo koristiti za pronalaženje fajlova resursa sa ikonama. Primetićete da sam definisao `Icon` svojstvo formulara (koristeći `aa.ico` fajl) kao i Icon svojstvo aplikacije (koristeći aa.ico fajl). Svaki put kada kliknete neku od Change kontrola, učitava se nova ikona iz spoljašnjeg fajla:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  with OpenDialog1 do
    if Execute then
    begin
      Application.Icon.LoadFromFile (Filename);
      Image1.Picture.LoadFromFile (Filename);
    end;
end;
```

Kada kliknete jednu od Remove kontrola, odgovarajuća ikona se jednostavno uklanja:

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  Application.Icon := nil;
  Image1.Picture := nil;
end;
```

Ovaj program možete koristiti da biste videli kako dva svojstva ikone utiču na ikonu minimizirane aplikacije. Neki primeri su dati na slici 19.3. Kada pokrenete aplikaciju Icons iz Explorera, Explorer će automatski upotrebiti ikonu aplikacije kada minimizirate formular, a ne ikonu glavnog formulara. To je zato što je ikona glavnog formulara sakrivena unutar korisničkog resursa koji opisuje formular (DFM fajl), dok se ikona aplikacije čuva u resursu ikone koji je povezan sa izvršnim fajlom na tradicionalan način.



SLIKA 19.3 *Neki efekti aplikacije Icons*

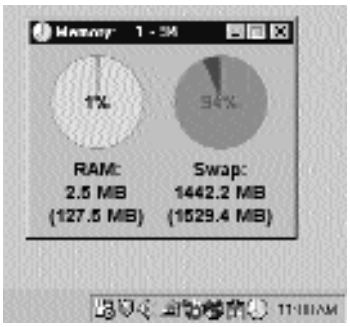
Upotreba polja ikona (Icon Tray) na Taskbaru

Windows 95 je predstavio novi način za prikazivanje sistemskih informacija: upotreba oblasti polja (tray — kutija, ladica) na Taskbaru. Zapravo, Windows omogućava programima da se izvršavaju samo kao ikone na ovoj oblasti. U donjem desnom uglu ekrana, blizu časovnika, postoji određeni prostor (Taskbar tray) koji možete upotrebiti da biste prikazali Vaše programe.

Samo jedna API funkcija se koristi za ovo prikazivanje, funkcija `Shell_NotifyIcon`. Ova funkcija je veoma jednostavna. Funkcija ima dva parametra: pokazivač na strukturu `TNotifyIconData` i zastavicu koja označava da li želite da dodate, uklonite ili izmenite ikonu. Polja strukture podataka uključuju veličinu (`cbSize`, koji se zapravo koristi za utvrđivanje verzije strukture), hendl prozora kojem ikona treba da šalje notifikacije (`hWnd`), broj poruke notifikacije (`uCallbackMessage`), identifikator ikone (`uID`), neke zastavice koje naznačavaju koja polja su obezbeđena, ikonu koja se prikazuje (`hIcon`) i Tooltip poruku (`szTip`).

Kada korisnik radi sa ikonom, Windows šalje nazad datom prozoru poruku definisanu programom, prosleđujući kao parametre akciju koju je korisnik izvršio nad ikonom (obično mišem) i ID date ikone. Ja sam koristio ove informacije o ikonama u primeru Mem3 koji u vreme izvršavanja možete videti na slici 19.4. Ovaj primer programa je koristan za prikazivanje statusa memorije; zasnovan je na API funkciji `GlobalMemoryStatus`, opisanoj u Poglavlju 18. Program koristi tri ikone za isticanje statusa memorije: zelenu kada još uvek imate nešto slobodne RAM memorije, žutu kada je RAM memorija popunjena ali ima još dosta mesta u swap fajlu, i crvenu kada je čak i swap fajl popunjen.

Pored praćenja obojenih ikona, možte preći preko njih pokazivačem miša da biste videli oblačić sa nekim detaljima; takođe možete kliknuti ikonu u polju Taskbara da biste otvorili prozor u kome se nalazi dosta dodatnih informacija, kao što je prikazano na slici 19.4. Program koristi neke napredne tehnike da bi se izbegli prikazivanje glavnog formulara prilikom pokretanja programa i stalno ažuriranje ikone. Ukoliko ste zainteresovani za ovakav tip programa, trebalo bi da pažljivo proučite izvroni kod.



SLIKA 19.4 *Primer Mem3 koristi dve oznake za prikazivanje detalja o statusu memorije i dodaje ikonu u polje na Taskbaru*

Upotreba kursora u Delphiju

Delphi podrška za kursore je velika, pa je potrebno daleko manje posla oko prilagođavanja kursora nego što je potrebno za prilagođavanje ikona. Na primer, Delphi uključuje veliki broj unapred definisanih kursora. Neki od njih su Windowsovi unapred određeni kursori, dok ostale obezbeđuje Delphi. Upotreba kursora u Delphiju je direktna; jednostavno upotrebite Object Inspector za selektovanje odgovarajuće vrednosti za svojstvo `Cursor` ili svojstvo `DragCursor` komponente. U Delphiju 5 možete čak videti oblik kursora u listi Object Inspectora, što izbor kursora čini intuitivnijim. Ukoliko je potrebno da odredite globalni kursor za celu aplikaciju za neko određeno vreme, možete upotrebiti svojstvo `Cursor` globalne komponente `Screen`. Naredni fragment koda demonstrira uobičajen način prikazivanja kursora koji sugeriše čekanje (peščani sat) za aplikaciju dok se izvršava dugačak zadatak:

```
Screen.Cursor := crHourglass;
try
  {time-consuming code would appear here}
finally
  Screen.Cursor := crDefault;
end;
```

Ovaj kod koristi obradu izuzetka da bi se osiguralo da čak i kada u izvršavanju nešto pođe naopako, može da se prikaže unapred određeni kursor. Svojstvo `Cursor` formulara i drugih komponenata i svojstvo `Cursor` objekta `Screen` su tipa `TCursor`. Ukoliko pogledate definiciju ovog tipa podataka u Delphijevom helpu, bićete iznenađeni. `TCursor` nije klasa već numerički tip. Tehnički, `TCursor` je celobrojna vrednost koju referencira niz kursor hendlova, koji se čuvaju u svojstvu `Cursors` (obratite pažnju na poslednje s) objekta `Screen`. Ovaj niz se takođe može koristiti za učitavanje novog kursora iz resursa aplikacije.

Upotreba resursa tabele stringova

Treći tip resursa kojim ćemo se pozabaviti u ovom poglavlju je tabela stringova (string table). Tabele stringova imaju važnu ulogu u Delphiju, a specifična podrška za tabele stringova je ugrađena u Object Pascal. Kada Vam je potrebna string konstanta u programu, umesto da je deklarišete u odeljku const, možete je jednostavno deklarisati u odeljku resourcestring:

```
resourcestring
  Text1 = 'This is some text';
```

Kada napišete ovu deklaraciju, Delphi automatski dodaje novi element u tabelu stringova aplikacije, koja će biti deo izvršnog fajla kao resurs. To znači da svaki put kada upotrebite string `Text1`, Delphi automatski dodaje kod za učitavanje stringa iz resursa tabele stringova. Ovaj kod ćemo videti malo kasnije. Efekat ovakvog pristupa je različito uređenje memorije koda i podataka programa, koje obično donosi korist jer operativni sistem obrađuje resurse na veoma efikasan način.

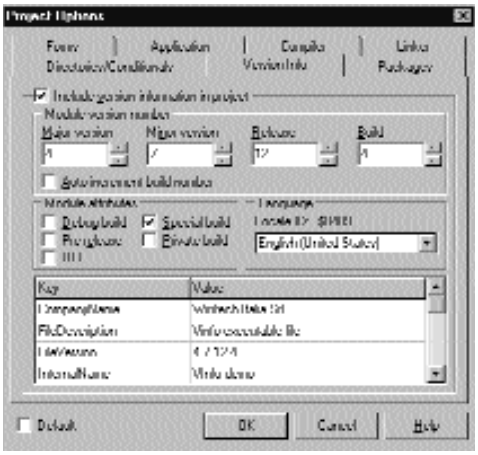
Drugi razlog za upotrebu resursa u Windows programima je pojednostavljivanje lokalizacije, ili translacija programa u neki drugi jezik (recimo iz engleskog u nemački). Da biste lokalizovali uobičajen program, trebalo bi da pretražite svaki fajl izvornog koda, pronađete tekst, prevedete tekst, a da zatim ponovo kompajlirate ceo program. Nasuprot tome, kada lokalizujete Windows aplikaciju koja koristi tabele stringova, samo prevedite tekst koji se nalazi u resursima, ponovo kompajlirajte samo resurse (veoma jednostavan proces) i zatim nove resurse povežite sa prethodno kompajliranim EXE kodom.

NAPOMENA

Proces lokalizacije u Delphiju ćemo razmatrati u jednom od kasnijih odeljaka ovog poglavlja koje u potpunosti posvećeno novom ITE-u, Integrated Development Environment. ■

Informacija o verziji

Poslednji tip resursa na koji ću obratiti pažnju su informacije o verziji. Jednostavno otvorite opcije projekta, pređite na stranu Version Info okvira za dijalog i unesite odgovarajuće vrednosti za broj verzije, naziv proizvoda, prava i druge informacije. Primer ovog okvira za dijalog možete videti na slici 19.5. Ove informacije o verziji Delphi projekta se dodaju standardnom RES fajlu (fajlu koji ima isti naziv kao i projekat), na koje se u opštem slučaju referiše iz izvornog koda projekta i koje se uključuju u izvršni fajl. Nemojte uklanjati ove reference iz izvornog koda DLL da biste mu dodali informacije o verziji.



SLIKA 19.5 *Strana Version Info okvira za dijalog Project Options*

Informacije o verziji su neophodne za DLL-ove i OLE servere (uključujući ActiveX kontrole) tako da instalacioni programi mogu odrediti da li već posedujete najnovije verzije DLL-ova. Bez ove tehnike rizikujete instaliranje starijih DLL-ova ili prepisivanje novijih verzija. Poznavanje informacija o verziji za DLL-ove je naročito važno ukoliko želite da načinite efektivnu upotrebu instalacionog programa (ili da napišete svoj program za instalaciju, koji poredi informacije o verziji fajlova koji već postoje na hard disku sa fajlovima koje instalirate).

Pošto možete naći dobar opis uloge informacija o verziji u Microsoftovoj dokumentaciji, ja ovaj opis neću ponavljati. Ono što želim da učinim umesto toga je da Vam pokažem jednostavnu upotrebu informacija o verziji u izvršnom programu. Ja sam dodao informacije o verziji na slici 19.5 programu VInfo i napisao sam kod koji izdvaja nešto od ovih informacija u memo komponentu kada korisnik klikne kontrolu Read Version Info formulara. Rezultat možete videti na slici 19.6. Problem je, kao što možete videti, u tome da je API koji se koristi za pristupanje informacijama o verziji daleko od jednostavnog.

SLIKA 19.6 *Informacije o verziji izdvojene iz primera VInfo*

Glavni API poziv je `GetFileVersionInfo`. Ova funkcija kao parametar zahteva naziv fajla, pokazivač na blok memorije u koji će se smestiti podaci, i veličinu ovog bloka memorije. Da biste alocirali blok memorije odgovarajuće veličine, program prvo može da pozove API funkciju `GetFileVersionInfoSize`. Sledi prvi deo obrade `OnClick` događaja:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  VInfoSize, DetSize: DWord;
  pVInfo, pDetail: Pointer;
begin
  Memo1.Lines.Clear;
  VInfoSize := GetFileVersionInfoSize (
    PChar (ParamStr (0)), DetSize);
  if VInfoSize > 0 then
  begin
    GetMem (pVInfo, VInfoSize);
    try
      GetFileVersionInfo (PChar (ParamStr (0)), 0,
```

```
          VInfoSize, pVInfo);
        ...
    finally
      FreeMem (pVInfo);
    end;
  end;
end;
```

Poslednji deo koda, unutar `finally` bloka, uklanja blok memorije. U međuvremenu, program koristi `pVInfo` pokazivač za pristupanje informacijama o verziji. Podacima pristupamo pozivanjem API funkcije `VerQueryValue`, koja kao parametre zahteva pokazivač na podatke, string koji sadrži putanju zahtevane informacije i pokazivač. Funkcija određuje ovaj pokazivač na zahtevani string ili strukturu podataka.

Prvi deo koda pristupa nepromenljivom delu informacija fajla, grupi zastavica i brojeva definisanih strukturom `TVSFixedFileInfo`. Sledi kod kojim se pristupa nekim podacima ove strukture (u preuzetim fajlovima možete pronaći dužu verziju):

```
// show the fixed information
VerQueryValue (pVInfo, '\', pDetail, DetSize);
with TVSFixedFileInfo (pDetail^) do
begin
  Memo1.Lines.Add (
    'Signature (should be invariably 0xFEEF04BD): '
    + IntToHex (dwSignature, 8));
  Memo1.Lines.Add ('Major version number: ' +
    IntToStr (HiWord (dwFileVersionMS)));
  if (dwFileFlagsMask and dwFileFlags
      and VS_FF_DEBUG) <> 0 then
    Memo1.Lines.Add ('Debug info included');
```

Drugi deo ovog koda čita neke stringove koji se nalaze unutar informacija o verziji programa. Svakom stringu bi trebalo da se pristupa zasebno, upotrebom API funkcije `VerQueryValue`. Evo kako možete da napišete jedan od ovih poziva:

```
VerQueryValue(pVInfo,
  '\StringFileInfo\040904E4\FileDescription', pDetail, DetSize);
Memo1.Lines.Add ('File Description: ' + PChar (pt2));
```

Ukoliko kod napišete na ovaj način, umetnućete lokalne informacije i informacije o skupu karaktera (`040904E4`) u kod. Mada ove informacije o jeziku možete odrediti na strani Project Options, Vaš kod bi trebalo da može da pročita trenutne vrednosti. Da biste ovo postigli, prvo bi trebalo da pročitate jezik iz informacije o verziji (što nije nimalo laka operacija sa pokazivačima) i da je zatim upotrebite za dalje procesiranje, kao u narednom kodu (deo metoda `Button1Click` primera VInfo):

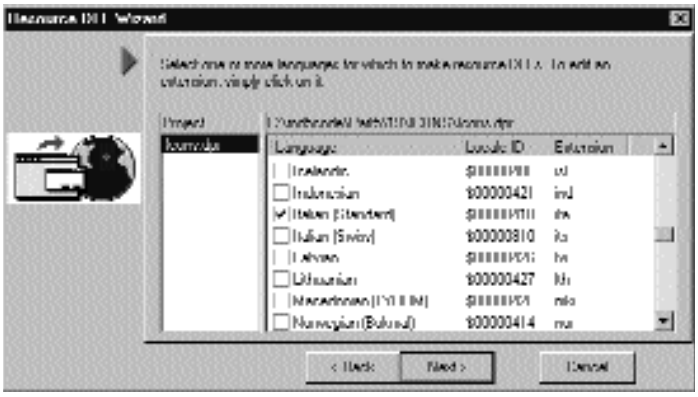
```
type
  TLangInfoBuffer = array [1..4] of SmallInt;
var
  pLangInfo: ^TLangInfoBuffer;
  strLangId: string;
begin
  // get the first language
```

```
VerQueryValue(pVInfo,
  '\VarFileInfo\Translation',
  Pointer(pLangInfo), DetSize);
strLangId := IntToHex (SmallInt (pLangInfo^ [1]), 4) +
  IntToHex (SmallInt (pLangInfo^ [2]), 4);
Memo1.Lines.Add ('Language: ' + strLangId);
// show some of the strings
strLangId := '\StringFileInfo\' + strLangId;
VerQueryValue(pVInfo, PChar(strLangId + '\FileDescription'),
  pDetail, DetSize);
Memo1.Lines.Add ('File Description: ' +
  PChar (pDetail));
```

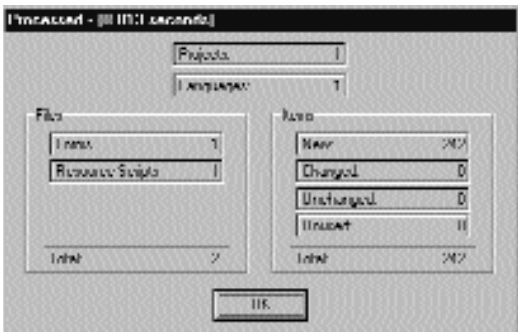
Integrisano okruženje za prevođenje

Jedna od potpuno novih karakteristika Delphija 5 je integrisano okruženje za prevođenje (Integrated Translation Environment), skraćeno ITE. U ovoj knjizi nema dovoljno prostora za detaljno razmatranje ovog složenog alata. Kao i obično, ja ću Vas samo usmeriti na neke njegove karakteristike i pokazaću Vam primer.

Da biste započeli proces prevođenja, možete upotrebiti Resource DLL Wizard koji se nalazi u okviru za dijalog File➥New ili upotrebiti komandu menija Project➥Language➥Add. Oba postupka otvaraju čarobnjaka tamo gde ste selektovali jedan od projekata aktivne grupe; zatim odaberite jezik (u svom primeru sam odabrao Italian standard — videti sliku 19.7), odaberite da li je ovo novo prevođenje ili ažuriranje postojećeg i kliknite kontrolu Finish. Ukoliko je ovo novi projekat, dobićete novi direktorijum i neku statistiku kao što je statistika koja je prikazana na slici 19.8.



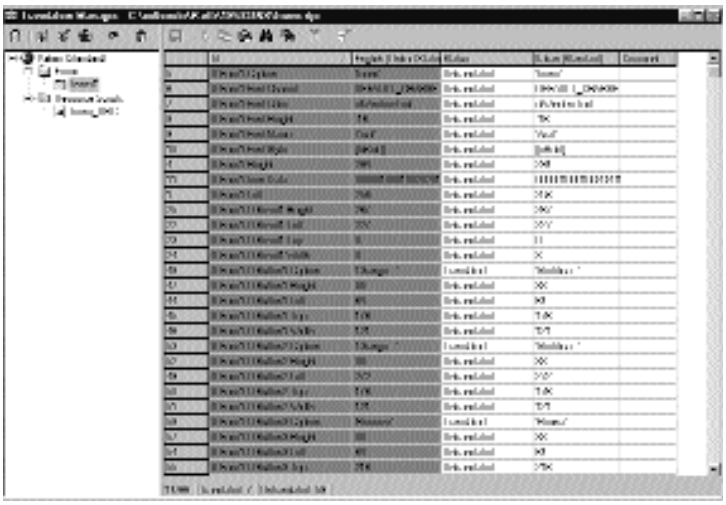
SLIKA 19.7 *Izbor jezika u Resource DLL Wizardu*



SLIKA 19.8 *Konačna statistika koju prikazuje Resource DLL Wizard posle dodavanja novog jezika projektu*

Kada je sve pravilno podešeno, možete početi rad sa Translation Managerom. To je okvir za dijalog u kome ITE prikazuje listu resursa, uključujući formulare i stringove, koji se mogu izmeniti za prevođenje. Ovaj prozor prikazuje sve elemente koje možete izmeniti prilikom prevođenja, prikazujući originalni tekst i tekst prevoda, i prati prethodne verzije i datume izmena. Možete filtrirati ovu tabelu i odabrati kolone koje želite da prikažete koristeći kontekst meni. Na slici 19.9 možete videti Translation Manager kada je selektovan formular (koji je preuzet iz mog italijanskog prevoda primera Icons).

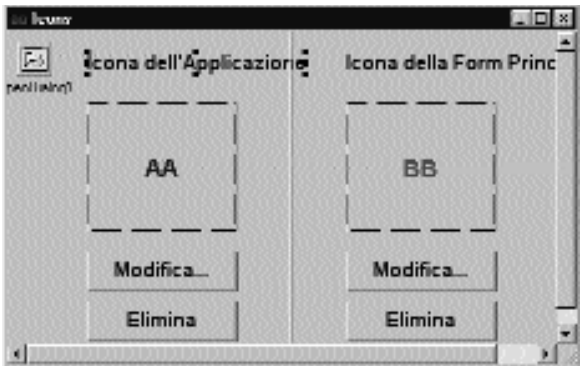
Translation Manager radi uz još jedan alat, Translation Repository, u kojem možete sačuvati standardni prevod čestih termina. Možete ručno ažurirati ove informacije (upotrebite komandu Tools➥Translation Repository) ili upotrebite komandu Repository➥Add strings to Repository iz kontekst menija Translation Managera. Možete upotrebiti još jednu komandu iz istog kontekst menija (komandu Get Strings from Repository) da biste automatski preveli sve termine koji su na raspolaganju. Primetićete da Repository može da obradi više prevoda za istu reč i da ima i druge napredne elemente.



SLIKA 19.9 *Novi Delphi 5 Translation Manager, koji je deo ITE-a*

Na slici 19.9 možete videti da se tekst stringova može izmeniti, kao što sam ja to učinio, ali takođe možete promeniti i druge parametre, kao što su pozicije i fontovi. Ovo je možda potrebno kada drugačija veličina prevedenih stringova utiče na korisnički interfejs. Naravno, samo pogledom na ove brojeve nije lako utvrditi da li je veličina i pozicija kontrola korektna. Ono što možete učiniti je da zatvorite Translation Manager, odaberete novi projekat koji je ITE dodao trenutnoj grupi projekta i pređete na formular, otvarajući ga za dizajniranje. To je ono što sam ja učinio da bih dobio prikaz sa slike 19.10, gde možete videti da je prevod na italijanski jezik prouzrokovao neke probleme sa veličinom oznaka. Interesantna karakteristika je da slobodno možete da izmenite prevedeni formular (ukoliko ne dodajete komponente na formular) pomeranjem kontrola, promenom njihovih fontova i tako dalje.

Zapravo, svaki put kada ponovo otvorite Translation Manager (selektovanjem glavnog projekta i zahtevanjem ne tako očigledne DLL komande Project➥Languages➥Update Resource), ponovo će se pročitati trenutne vrednosti iz DFM fajlova (iz originalnih i prevedenih) i prema tim informacijama će se osvežiti struktura Translation Managera. Na ovaj način, ukoliko ažurirate originalni formular, nećete morati da ga ponovo prevedete, već samo da obezbedite prevod za nove elemente. Istovremeno možete da izmenite prevedeni DFM fajl i vidite izmene u sistemu prevođenja. Isto sam i ja učinio u svom primeru.



SLIKA 19.10 *Kada prevodite naslove, neki od njih mogu postati veći, što dovodi do problema u programu*

Kada ste kompajlirali prevedeni projekat, koji je zapravo DLL, možete upotrebiti komandu Project➥Languages➥Set Active da biste ga aktivirali, tako da će izvršavanje projekta u debageru učitati aktivnu ekstenziju jezika. Ovo je samo test; obično će biti potrebno da samo pokrenete glavni izvršni fajl i on će automatski upotrebiti prevedeni DLL koji odgovara trenutnim selektovanim lokalnim vrednostima (preko pomoćnog programa Regional Settings Control Panela).

Kako ovo funkcioniše i šta se dešava dalje od očiju? Resource DLL Wizard kreira DLL projekat koji uključuje prevedeni DFM i stringove. DLL sadrži samo resurse, ne i kopiju kompajliranog koda, i sadrži proširenje koje odgovara kodu od tri slova kojim se identifikuje lokalni jezik, ITA u ovom slučaju. Ova struktura je vidljiva u izvornom kodu projekta:

```
library Icons;
{ITE} {$R 'IconsF.dfm' Form1:TForm}
  {DFMFileType} {IconsF.dfm}
{ITE} {$R 'Icons_DRC.res' 'Icons_DRC.rc'}
```

```
  {RCFileType} {Icons_DRC.rc}
{$E ita}
begin
end.
```

Dve *$R* direktive određuju resurse koji se uključuju u projekat, a za njima slede komentari koji su neophodni za ITE (ne smete ih menjati). Direktiva *$E* određuje proširenje izvršnog fajla. ITE kreira poddirektorijum za svako lokalno podešavanje tako da nazivi fajlova ne mogu dovesti do konflikta.

Kada kompajlirate DLL resurs, njegov izlaz se smešta u roditeljski direktorijum, isti onaj gde je smešten projekat, tako da se odmah može koristiti. Kada pokrenete ovaj izvršni fajl, VCL će učitati resurse bilo iz glavnog EXE fajla ili DLL fajla koji odgovaraju trenutnim regionalnim vrednostima.

SAVET

Da biste aktivirali prevedeni program, potrebno je da zatvorite onaj koji se trenutno izvršava, promenite regionalna podešavanja i izvršite program. Takođe je moguće promeniti jezik i tokom izvršavanja, ponovnim učitavanjem formulara bez zaustavljanja programa. Ipak, trenutni unos korisnika će biti izgubljen jer formulari treba da se kreiraju iz početka. Ukoliko želite da istražite ovaj dinamički pristup, pogledajte primer RichEdit koji je deo Delphi demo programa, naročito globalne funkcije u jedinici ReInit.pas. Dodavanjem ove jedinice Vašim programima, moći ćete da dobijete dinamičku promenu jezika koju ovaj program demonstrira. ■

Štampanje

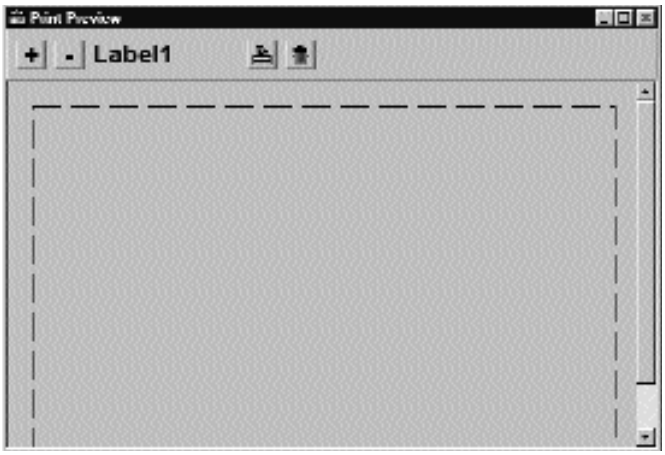
Delphi podržava štampanje na brojne načine. Formulari mogu da odštampaju svoje grafičke prikaze (pogledajte metod `Print` i svojstvo `PrintScale` klase `TForm`), a neke komponente imaju direktnu podršku za štampanje, kao što je kontrola RichEdit. Za sve jednostavne operacije upotrebljavaćete promenljivu `Printer` za manipulisanje štampačem iz Delphi programa. Zapravo, `Printer` je naziv globalne funkcije; ona kao rezultat daje objekat klase `TPrinter`, koji je definsan u jedinici Printers.

Objekat koji dobijate kao rezultat možete koristiti u funkciji `Printer` za pristupanje globalnim svojstvima koja se odnose na štampač, kao što je spisak instaliranih drajvera ili fontova štampača. Ipak, ključno svojstvo je `Canvas`. Slike štampača možete koristiti na isti način na koji koristite slike formulara, to jest, možete štampati tekst, grafiku i sve ostalo. Da biste upotrebili ove slike, potrebno je da pozovete metod štampača `BeginDoc` (da biste započeli posao štampanja), da upotrebite metode slika (da biste proizveli izlaz) i da zatim pozovete metod `EndDoc` (da biste štampaču poslali izlaz). Kao alternative možete pozvati metod `Abort` da biste odbacili posao štampanja, ili pozvati metod NewPage da biste poslali izlaz štampaču i započeli rad na novoj strani.

Print Preview grafika

Naš prvi primer sa globalnim objektom štampača (upotrebom funkcije `Printer`) je jednostavna aplikacija koju možete upotrebiti za štampanje bitmapa. U osnovi, ovo je proširenje primera TabOnly koji je prikazan u Poglavlju 8. Taj primer je koristio komponentu TabControl da bi korisnicima omogućio da pretražuju niz bitmapa. Kao što je prikazano na slici 19.11, primer PrintBmp prikazuje izgled formulara koji sadrži paletu alata sa četiri kontrole na vrhu i kompo-

nentu ScrollBox koja sadrži komponentu Image. Ukoliko je slika veća od formulara, možete da upotrebite komponentu ScrollBox da biste skrolovali sliku, a da time ne utičete na paletu alata.



SLIKA 19.11 *Print Preview formulara primera PrintBmp u vreme dizajniranja*

Ovaj okvir za dijalog je otvoren preko aplikacijske komande File➥Print. Formular Vam omogućava da uporedite veličinu rezultujuće bitmape sa odštampanom stranom (naznačena je veličinom komponente za sliku) i promenite proporcije, ukoliko je neophodno, da biste povećali veličinu.

NAPOMENA

Promena veličine slike utiče kako na prikaz na ekranu u formularu, tako i na štampani izlaz. Razlog za skaliranje bitmape pre nego što se odštampa je taj što se bitmape, kada se odštampaju po njihovom standardnom broju piksela po inču, na papiru prikazuju kao veoma male. ■

Kod se zasniva na metodu `StretchDraw` klase `TCanvas`, koji sam upotrebio za generisanje preliminarnog prikaza bitmape i za štampanje bitmape. Metod `StretchDraw` sadrži dva parametra: pravougaonik koji naznačava oblast štampanja i grafički objekat (sliku koja se štampa). Pogledajmo sada deo koda. Glavni formular reaguje na komandu Print inicijalizovanjem i izvršavanjem formulara Preview:

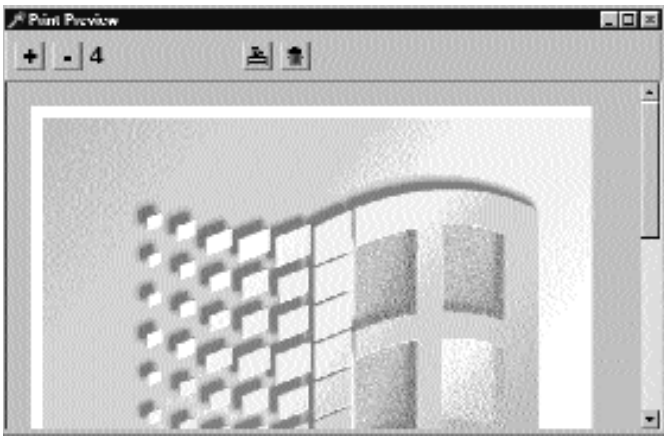
```
procedure TForm1.Print1Click(Sender: TObject);
begin
  {double-check whether an image is selected}
  if Image1.Picture.Graphic <> nil then
  begin
    {set a default scale, and start the preview}
    PreviewForm.Scale := 2;
    PreviewForm.SetPage;
    PreviewForm.DrawPreview
    PreviewForm.ShowModal;
  end;
end;
```

Test na početku se može izostaviti, jer je element menija Print neaktivan sve dok se ne odabere fajl sa slikom, ali ovaj test osigurava da je fajl selektovan u bilo kom slučaju. Ovaj kod određuje vrednost za javno polje objekta `PreviewForm` (polje `Scale`), poziva dva metoda ovog formulara (metode `SetPage` i `DrawPreview`) i na kraju ga prikazuje kao prioritetni formular. Na slici 19.12 je prikazan primer formulara Print Preview u vreme izvršavanja.

Metod `SetPage` određuje veličinu komponente Image formulara Print Preview, koristeći veličinu odštampane strane:

```
procedure TPreviewForm.SetPage;
begin
  Image1.Width := Printer.PageWidth div 5;
  Image1.Height := Printer.PageHeight div 5;
  {output the scale to the toolbar}
  Label1.Caption := IntToStr (Scale);
end;
```

Veličina strane se deli sa pet da bi se smestila u razumnu površinu ekrana. Vi možete da koristite parametar umesto ove stalne vrednosti da biste dodali karakteristiku zumiranja. Ipak, izgleda da je pomalo zbunjujuće da imate jednu kontrolu za povećavanje veličine štampane slike, a drugu za njeno povećavanje samo za prikaz, te sam odlučio da izostavim mogućnost zumiranja.



SLIKA 19.12 *Formular Print Preview primera PrintBmp, kada je glavni formular programa u pozadini*

Srž koda formulara za preliminarni prikaz je u metodu `DrawPreview`, koji ima tri odeljka. Na početku kod izračunava odredišni pravougaonik, ostavljajući marginu od 10 piksela, skalirajući sliku i koristeći stalni faktor zumiranja čija je vrednost 5 (kao što možete videti u narednom listingu). Drugi korak uklanja staru sliku, koja se još uvek nalazi na ekranu, iscrtavanjem belog pravougaonika preko slike. Treći korak je poziv metoda slike `StretchDraw`, koristeći pravougaonik koji je prethodno izračunat i aktuelnu sliku iz komponente Image glavnog formulara (`Form1.Image1.Picture.Graphic`). Da biste pristupili ovoj informaciji, potrebno je da dodate klauzulu uses u odeljak implementation koda, tako da se referiše na jedinicu `Viewer` (koja deklariše klasu `TForm1`). Sledi kod metoda `DrawPreview`:

```
procedure TPreviewForm.DrawPreview;
var
  Rect: TRect;
begin
  {compute the rectangle for the bitmap preview}
  Rect.Top := 10;
  Rect.Left := 10;
  Rect.Right := 10 +
    (Form1.Image1.Picture.Graphic.Width * Scale) div 5;
  Rect.Bottom := 10 +
   (Form1.Image1.Picture.Graphic.Height * Scale) div 5;
  {remove the current image}
  Image1.Canvas.Pen.Mode := pmWhite;
  Image1.Canvas.Rectangle (0, 0,
    Image1.Width, Image1.Height);
  {stretch the bitmap into the rectangle}
  Image1.Canvas.StretchDraw (Rect,
    Form1.Image1.Picture.Graphic);
end;
```

Sav ovaj kod se izvršava samo da bi se inicijalizovao formular. Početni kod je podeljen na dva metoda, ali samo zato što će se `DrawPreview` kasnije ponovo pozivati. Kada je inicijalizacija završena, a prioritetni formular se prikazuje, korisnik može kliknuti četiri kontrole palete alata da bi promenio veličinu slike, štampao je i prešao na drugu sliku.

Dva metoda za promenu veličine slike su jednostavna jer samo određuju novu vrednost skaliranja i pozivaju proceduru `DrawPreview` za ažuriranje slike:

```
procedure TPreviewForm.ScalePlusButtonClick(
  Sender: TObject);
begin
  Scale := Scale * 2;
  Label1.Caption := IntToStr (Scale);
  DrawPreview;
end;
```

Metod `PrintButtonClick` formulara za preliminarno prikazivanje slike gotovo da je klon metoda `DrawPreview`. Jedine razlike su u tome da se odredišni pravougaonik ne zumira i da se bitmapa šalje štampaču u novom dokumentu (novoj strani):

```
procedure TPreviewForm.PrintButtonClick(Sender: TObject);
var
  Rect: TRect;
begin
  {compute the rectangle for the printer}
  Rect.Top := 10;
  Rect.Left := 10;
  Rect.Right := 10 +
    (Form1.Image1.Picture.Graphic.Width * Scale);
  Rect.Bottom := 10 +
    (Form1.Image1.Picture.Graphic.Height * Scale);
  {print the bitmap}
  Printer.BeginDoc;
  try
```

```
    Printer.Canvas.StretchDraw (Rect,
      Form1.Image1.Picture.Graphic);
    Printer.EndDoc;
  except
    Printer.AbortDoc;
    raise;
  end;
end;
```

NAPOMENA

Kada Vaš program iscrtava po formularu, može se adaptirati tako da proizvede isti prikaz direktno na štampaču. Isti kod može dati izlaz za generičku sliku i posle promene koordinatnog sistema. Ovo je demonstrirano primerom Shapes u bonus poglavlju koje razmatra grafiku (posetite Sybexov web sajt na adresi `www.sybex.com`). ■

Štampanje teksta

Postoje slučajevi kada je potrebno da neki tekst direktno odštampate što je brže moguće, bez potrebe za dodatnom grafikom. Kada je to slučaj, možete se osloniti na podršku direktnom štampanju koju nude tekst fajlovi. Kada ste kreirali fajl koji čuva tekst, taj fajl možete pridružiti štampaču i u njega zapisivati tekst. Pogledajte naredni kod, koji je deo primera QrNav:

```
procedure TPreviewForm.PrintButtonClick(Sender: TObject);
var
  PrintFile: TextFile;
begin
  {assigning the printer to a file}
  AssignPrn (PrintFile);
  Rewrite (PrintFile);
  try
    {set the font of the form, and output each element}
    Printer.Canvas.Font := Font;
    Writeln (PrintFile, Label1.Caption,
      ' ', DBEdit1.Text);
    Writeln (PrintFile, Label2.Caption,
      ' ', DBEdit2.Text);
    Writeln (PrintFile, Label3.Caption,
      ' ', DBEdit3.Text);
  finally
    {close the printing process}
    System.CloseFile (PrintFile);
  end;
end;
```

Ova obrada događaja štampa tekst nekoliko polja za izmene i tekst odgovarajućih oznaka. Kao što iz izvornog koda možete videti, program koristi iste tehnike za štampanje celokupnog sadržaja tabele baze podataka.

NAPOMENA

Ovaj tip podrške štampanju se još uvek oslanja na Windows drajvere za štampač, koji mogu generisati samo grafički izlaz. Ukoliko želite da zaista budete brzi, možete upotrebiti `Escape` API da biste direktno upravljali svojim štampačem pomoću Escape komandi. Neki matrični štampači mogu veoma brzo da štampaju kada koriste neprekidni papir, a ovo može biti zgodan način za iskorišćenje njihove maksimalne brzine. ■

Komponente QuickReport

Profesionalna verzija Delphija sadrži QuickReport, kolekciju komponenata za pravljenje izveštaja koje su čvrsto integrisane u Delphi i licencirane za Borlandov QSD AS, Norway. Slične Delphi komponente su na raspolaganju kod nezavisnih programera, ali ću se ja pozabaviti ovom komponentom jer će ubrzo biti dostupna većini Delphi programera.

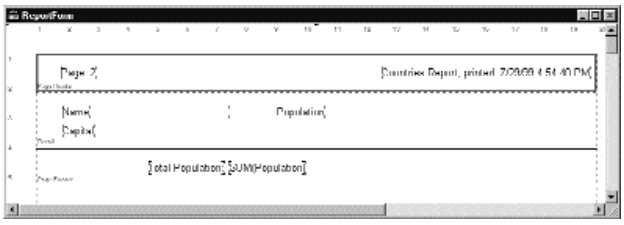
NAPOMENA

Alternative skupu komponenata za izveštaje QuickReport su ReportPrinter, ReportBuilder (nekada je bio poznat kao Piparti), ACE Reporter i mnoge druge. ■

QuickReport koristi formular za vizuelnu izradu izveštaja na sličan način na koji Vi izrađujete uobičajene formulare. Ipak, Vi ćete ovaj formular za izveštaje koristiti samo za izradu iveštaja; on se zapravo nikada ne prikazuje na ekranu u vreme izvršavanja. Da biste odštampali ili prikazali izveštaj, možete pozvati metode `Print` ili `Preview` komponente QuickReport, koju smeštate na svaki formular za izveštaj (smeštanjem komponente QuickReport na formular, Vi formular pretvarate u formular za izveštaj).

Upotrebom komponente QuickReport, izveštaj se konstruiše iz grupa (bands) ili horizontalnih oblasti sa informacijama. Grupu možete koristiti za prikazivanje podataka, obezbeđivanje zaglavlja i podnožja na svakoj štampanoj strani i za druge specijalne informacije. Da biste izradili izveštaj, jednostavno smestite komponentu QuickReport na sekundarni formular (ne na glavni formular aplikacije), dodajte jednu ili više grupa, a zatim u te grupe neke QuickReport komponente za izveštaje koje prepoznaju podatke, koje povezujete sa Delphi izvorom podataka na uobičajen način. Podaci se mogu dobiti iz jedne ili više tabela ili upita, kao kod standardnih komponenata za pristup podacima.

Upotreba komponenata QuickReport je pokazana u primeru QrNav koji je pomenut u prethodnom odeljku. Sekundarni formular programa, formular izveštaja, sadrži komponentu QuickReport i tri komponente QRBand, kao što je prikazano na slici 19.13.



SLIKA 19.13 *Formular koji je upotrebljen za izradu izveštaja primera QrNav u vreme dizajniranja. Ovaj formular koriste komponente za izveštaj koje se nalaze na formularu, ali se formular nikada ne prikazuje na ekranu u vreme izvršavanja*

Jedno od ključnih svojstava komponente QRBand je `BandType`, koje se koristi za naznačavanje uloge grupe u izveštaju. U ovom primeru, prva grupa je tipa `rbPageHeader`, druga grupa je tipa `rbDetail`, a treća grupa je tipa `rbPageFooter`. Ostali tipovi grupa koje možete upotrebiti su `rbTitle`, koja se smešta pre ili posle zaglavlja prve strane; `rbSummary`, koja se štampa samo na kraju izveštaja; `rbGroupHeader` i `rbGroupFooter` za grupe definisane specijalnom komponentom QRGroup; `rbColumnHeader` za izveštaje sa više kolona; i nekoliko drugih.

U primeru sam dve QRSysData komponente smestio u prvu grupu (zaglavlje strane). Ove komponente prikazuju broj strane, datum i vreme, a njihova svojstva `Text` sadrže opis. Možete štampati mnoge druge tipove sistemskih informacija upotrebom ove komponente, kao što je naznačeno svojstvom `Data`. Ja sam takođe odredio borduru za uokviravanje grupe upotrebom svojstva `Frame`.

Druga grupa sadrži stvarne podatke iz baze podataka. Ova grupa se replicira na izveštaju za svaki slog koji se javlja u izvoru podataka, tako da morate da naznačite skup podataka za svaku komponentu izveštaja. U ovom slučaju sam koristio `Table1` iz glavnog formulara programa (pošto sam odabrao File➥Use Unit). Ovaj isti skup podataka je takođe povezan i sa komponentama QRDBText koje su postavljene na drugu grupu.

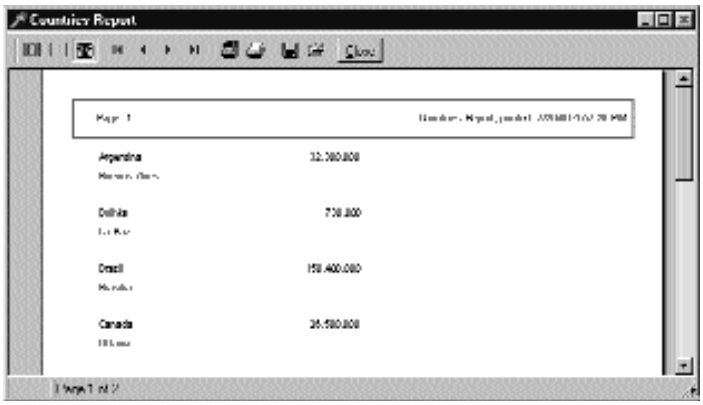
SAVET

Pored svojstva `DataField` koje dele sa standardnim Delphi komponentama koje prepoznaju podatke, komponente za izveštaje takođe imaju i neke mogućnosti formatiranja. Ukoliko pokušate da odredite vrednost `'###,###,###'` za svojstvo `Mask`, brojevi će biti prikazani sa separatorima za hiljade. ■

U poslednju grupu sam dodao komponentu QRExpr da bih prikazao ukupnu populaciju zemalja koje se prikazuju na izveštaju (zapravo iz svih slogova koji se prikazuju na strani). Ova komponenta može da izvrši složena izračunavanja. Najjednostavniji pristup je upotreba svojstva `Expression` za naznačavanje vrste operacije (kao što su sum, min, max, average ili count) i polja nad kojim se obavlja operacija (kao što je polje Population). Svojstvo `Expression` sadrži specijalni editor koji možete upotrebiti za kreiranje izraza, umesto da ih unosite. Ne zaboravite da postavite svojstvo `Master` na komponentu izveštaja, komponentu `QuickRep1`, jer je to jedini način za povezivanje izračunate vrednosti sa odgovarajućim skupom podataka.

Kada ste dizajnirali izveštaj, možete ga testirati tako što ćete dva puta kliknuti komponentu za izveštaj. Na ovaj način se prikazuje preliminarni izgled izveštaja, koji možete direktno upotrebiti za štampanje izveštaja, a da čak ne morate da kompajlirate program. Isti preliminarni prikaz možete dobiti u vreme izvršavanja (videti sliku 19.14) uključivanjem sledećeg koda u kod glavnog formulara:

```
procedure TNavigator.ReportButtonClick(Sender: TObject);
begin
  ReportForm.QuickReport1.Preview;
end;
```



SLIKA 19.14 *Preliminarni prikaz formulara primera QrNav koji se zasniva na komponenti QuickReport*

Manipulisanje fajlovima

Jedna od neobičnosti jezika Pascal u poređenju sa drugim programskim jezicima je njegova ugrađena podrška za fajlove. Jezik sadrži ključnu reč `file`, koja je specifikator tipa, kao što su to `array` ili `record`. Ključnu reč `file` možete upotrebiti za definisanje novog tipa podataka, a zatim novi tip podataka možete upotrebiti za deklarisanje novih promenljivih:

```
type
  IntFile: file of Integers;
var
  IntFile1: IntFile;
```

Takođe je moguće upotrebiti ključnu reč `file` bez navođenja tipa podataka, da biste naznačili fajl bez tipa. Alternativno, možete upotrebiti tip podataka `TextFile`, definisan u jedinici System, da biste deklarisali fajlove koji sadrže ASCII karaktere. Za svaki tip fajla postoje unapred određene rutine.

Kada ste deklarisali promenljivu fajla, možete je pridružiti pravom fajlu iz sistema fajlova upotrebom metoda `AssignFile`. Sledeći korak je obično poziv metoda `Reset` za otvaranje fajla na početku za čitanje, `Rewrite` za kreiranje novog fajla, ili `Append` (koji se odnosi samo na fajlove tipa TextFile) za dodavanje novih elemenata na kraj fajla, a da se elementi koji već postoje ne uklone. Kada se obave operacije ulaza ili izlaza, trebalo bi da pozovete metod `CloseFile`. Ova operacija bi tipično trebalo da se obavi unutar `finally` bloka, da bi se izbeglo da se fajl ostavi otvoren u slučaju da kod kojim se manipuliše fajlom generiše izuzetak.

Podrška fajlovima u Delphi komponentama

Pored standardne podrške fajlovima u jeziku Pascal, Delphi sadrži i brojne druge opcije za manipulisanje fajlovima. Nekoliko komponenata sadrži metode za čuvanje ili učitavanje njihovog sadržaja iz fajla (recimo tekst fajla ili fajla koji sadrži bitmapu), a postoje i druge specifične klase za manipulisanje fajlovima. Mnoge klase komponenata sadrže metode `SaveToFile` i `LoadFromFile`. U ovoj knjizi smo ove metode koristili za klase `TBitmap`, `TPicture` i `TStrings` (koristili smo ih za `TMemo`, `TListBox` i mnoge druge klase komponenata). Ovi metodi su takođe

na raspolaganju i za neke komponente koje prepoznaju podatke (`TBlobField`, `TMemoField` i `TGraphicField`), za ostale grafičke formate (`TGraphic`, `TIcon` i `TMetaFile`), za OLE kontejnere (povezivanje i ugnežđenje objekata) i za TreeView i druge Windows uobičajene kontrole.

Slični metodi su na raspolaganju za klasu `TMediaPlayer`. Ovi metodi su nazvani `Open` i `Save`, i imaju nešto drugačiju sintaksu i značenje od metoda `LoadFromFile` i `SaveToFile` koji su im parnjaci. Još jedna klasa za falove koju ćemo koristiti kasnije u ovom poglavlju je klasa `TIniFile`. Ova klasa implementira upravljanje Windows inicijalizacionim fajlovima ili bilo kojim fajlom koji koristi isti format. Nove teme koje ćemo razmatrati u narednim odeljcima su komponente fajl sistema, komponente za usmeravanje i komponente koje implementiraju stalnost objekta.

Komponente fajl sistema

Delphi komponente sistema fajlova se nalaze na System strani Components Palette: `TDirectoryListBox`, `TDriveComboBox`, `TFileListBox` i `TFilterComboBox`. Ove komponente slede staromodni korisnički interfejs Windowsa 3.1, te zbog toga nisu izuzetno korisne. Ipak, ista jedinica FileCtrl koja definiše ove komponente sadrži i tri interesantne rutine:

- rutinu `DirectoryExists`, koja se koristi za proveru da li direktorijum postoji;
- rutinu `ForceDirectories`, koja odjednom kreira nekoliko direktorijuma;
- rutinu `SelectDirectory`, koja prikazuje standardni Delphi okvir za dijalog za odabir direktorijuma.

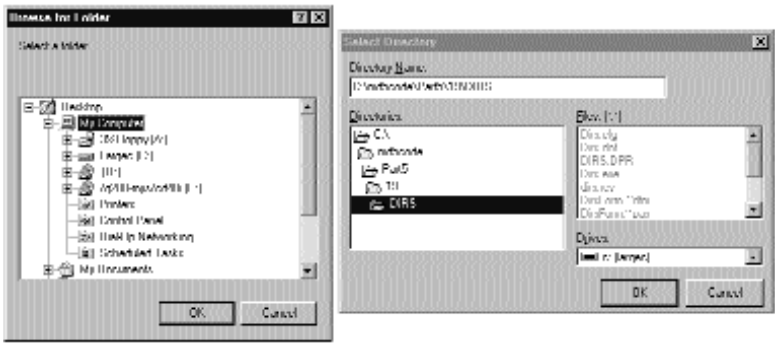
Primer Dirs demonstrira upotrebu ovih malo poznatih rutina. U izvornom kodu možete videti odgovarajuće pozive. Zapravo, Windows školjka sadrži veliki broj sličnih ali naprednijih rutina i nudi brojne okvire za dijalog, uključujući i okvir za dijalog za selektovanje direktorijuma. Za početak možda želite da proverite Windows API Namespace Functions grupom Shell. Postoji mnogo dobrih stvari u toj grupi, a mnoge i nisu dokumentovane!

U primeru Dirs sam načinio samo jednostavan korak na tu stranu. Kontrola poziva API funkciju `ShBrowseForFolder` koja prikazuje sistemski okvir za dijalog, koji se koristi za pronalaženje direktorijuma (ili štampača ili kompjutera, u zavisnosti od parametara). Sledi kod:

```
uses
  ShlObj;
procedure TForm1.btnBrowseClick(Sender: TObject);
var
  bi: TBrowseInfo;
  pidl: pItemIdList;
  strpath: string;
begin
  bi.hwndOwner := Handle;
  bi.pidlRoot := nil;
  bi.pszDisplayName := '';
  bi.lpszTitle := 'Select a folder';
  bi.ulFlags := bif_StatusText;
  bi.lpfn := nil;
  bi.lParam := 0;
  pidl := ShBrowseForFolder (bi);
  SetLength (strPath, 100);
```

```
  ShGetPathFromIdList (pidl, PChar(strPath));
  Edit1.Text := strPath;
end;
```

Efekat ovog koda možete videti na slici 19.15, pored Delphijevog okvira za dijalog koji je generisan pozivom `SelectDirectory`.



SLIKA 19.15 *Windowsov okvir za dijalog Browse For Folder i Delphijev okvir za dijalog Select Directory koji su aktivirani primerom Dirs*

SAVET

Kada je potrebno da radite sa fajlovima i direktorijumima, možda ćete želeti da proverite novu klasu `TMask` koja je predstavljena u Delphiju 5. Ova nova klasa je deklarisana u jedinici Masks (koju ne treba pomešati sa starijom jedinicom Mask koja definiše maske za editovanje). Klasa `TMask` Vam omogućava da izvršite poređenje po šablonu upotrebom džoker karaktera (standardni karakteri * i ?), uporedite vrednosti sa skupom maski i proverite opsege. Pogledajte Delphijev help fajl i informacije koje se odnose na `TMask.Create`. ■

Usmeravanje podataka

Još jedna tema koju vredi proučiti je Delphijeva podrška tokovima fajlova (file strams). VCL definiše apstraktnu klasu `TStream` i njene brojne potklase. Roditeljska klasa, klasa `TStream`, sadrži samo nekoliko svojstava, ali takođe sadrži brojne metode koje možete upotrebiti za čuvanje ili učitavanje podataka.

Kreiranje instance `TStream` nema nikakvog smisla jer je ova klasa apstraktna i ne obezbeđuje nikakvu direktnu podršku za čuvanje podataka. Umesto toga, možete upotrebiti jednu od izvedenih klasa za učitavanje podataka iz fajla ili za čuvanje podataka u fajlu, BLOB polju, priključku (socket) ili memorijskom bloku. Upotrebite klasu `TFileStream` kada želite da radite sa fajlom, prosleđujući naziv fajla i neke opcije metodu `Create`. Upotrebite klasu `TMemoryStream` za manipulisanje tokom u memoriji, a ne za manipulisanje fajlom. Ipak, ova klasa sadrži specijalni metod za kopiranje njenog sadržaja u neki tok ili iz njega (stream), koji može biti tok fajla. Kreiranje i upotreba toka fajla je jednostavna koliko i kreiranje promenljive tipa koji je izveden iz klase `TStream`:

```
var
  S: TFileStream;
begin
  if OpenDialog1.Execute then
  begin
    S := TFileStream.Create (OpenDialog1.FileName,
      fmOpenRead);
    try
      {use the stream S ...}
    finally
      S.Free;
    end;
  end;
end;
```

Kao što iz ovog koda možete videti, metod `Create` za tokove fajlova sadrži dva parametra: naziv fajla i zastavicu koja naznačava zahtevani mod pristupa. U ovom slučaju, mi želimo da pročitamo fajl, te smo koristili zastavicu `fmOpenRead` (ostale zastavice koje možete upotrebiti su dokumentovane u Delphi helpu). Tokovi se mogu koristiti umesto tradicionalnih Pascal fajlova, mada na početku mogu biti manje intuitivni u upotrebi. Velika prednost tokova je u tome da su oni veoma kompatibilni, tako da možete raditi sa memorijskim tokovima i zatim ih sačuvati u fajlu, ili možete obaviti suprotne operacije. Ovo može biti način da povećate brzinu programa koji mnogo koristi fajlove. Evo dela koda, funkcije za kopiranje fajlova, da biste dobili ideju kako možete koristiti tokove:

```
procedure CopyFile (SourceName, TargetName: String);
var
  Stream1, Stream2: TFileStream;
begin
  Stream1 := TFileStream.Create (SourceName, fmOpenRead);
  try
    Stream2 := TFileStream.Create (TargetName,
      fmOpenWrite or fmCreate);
    try
      Stream2.CopyFrom (Stream1, Stream1.Size);
    finally
      Stream2.Free;
    end
  finally
    Stream1.Free;
  end
end;
```

Druga važna upotreba tokova (kako tokova fajlova tako i tokova memorije) je za obradu BLOB polja baze podataka ili za direktnu obradu drugih velikih polja. Zapravo, Vi možete izvesti takve podatke u tok ili ih pročitati jednostavnim pozivanjem metoda `SaveToStream` i `LoadFromStream` klase `TBlobField`.

Clipboard

U Delphiju je podrška za Clipboard data na dva načina:

- Neke komponente imaju specifične metode koji se odnose na Clipboard. Na primer, `TMemo`, `TEdit` i `TDBImage`, među mnogim komponentama, imaju metode `CopyToClipboard`, `CutToClipboard` i `PasteFromClipboard`.
- Postoji i globalni objekat `Clipboard` klase `TClipboard`, koji sadrži veliki broj specifičnih Clipboard funkcija. Za potpunu podršku Clipboardu neophodna je upotreba objekta `Clipboard` koji je definisan u jedinici `ClipBrd`.

Program može koristiti objekat `Clipboard` da bi proverio da li Clipboard trenutno sadrži podatke zahtevanog formata, recimo, tekst ili bitmapu, upotrebom metoda `HasFormat`. Pošto Clipboard takođe može da sadrži više verzija istih podataka, ponekad je korisno izlistati sve moguće formate. Konačno, možete upotrebiti globalni objekat za smeštanje podataka na `Clipboard`, kada ova funkcija nije direktno podržana drugim komponentama. Objekat Clipboard se, takođe, može upotrebiti za otvaranje Clipboarda i za kopiranje podataka u različitim formatima. Ovo je jedini slučaj u kome je potrebno da otvorite i zatvorite Clipboard u Delphiju — nešto što je takođe neophodno kada direktno koristite Windows API.

Kopiranje i premeštanje teksta

Već smo videli primer upotrebe Clipboarda u Poglavlju 7. Primer Actions je koristio komponentu ActionList za implementiranje tipičnih operacija Cut, Copy i Paste nad tekstom Memo kontrole. U tom slučaju, interakcija sa Clipboardom i sa ažuriranjem korisničkog interfejsa je bila obrađivana unapred određenim akcijama. U Poglavlju 13, primer ListText jedemonstrirao kako da napišete slične akcije za liste, upotrebom globalnog objekta `Clipboard`.

Delphi help fajl dokumentuje pet različitih formata za metod HasFormat: `CF_TEXT`, `CF_PICTURE`, `CF_BITMAP` i `CF_METAFILE`. Ove formate tipično koriste Delphi i VCL komponente. Windows API definiše mnogo više formata, uključujući i sledeće formate:

`CF_BITMAP`	`CF_DSPMETAFILEPICT`
`CF_OWNERDISPLAY`	`CF_SYLK`
`CF_DIB`	`CF_DSPTEXT`
`CF_PALETTE`	`CF_TEXT`
`CF_DIF`	`CF_METAFILEPICT`
`CF_PENDATA`	`CF_TIFF`
`CF_DSPBITMAP`	`CF_OEMTEXT`
`CF_RIFF`	`CF_WAVE`

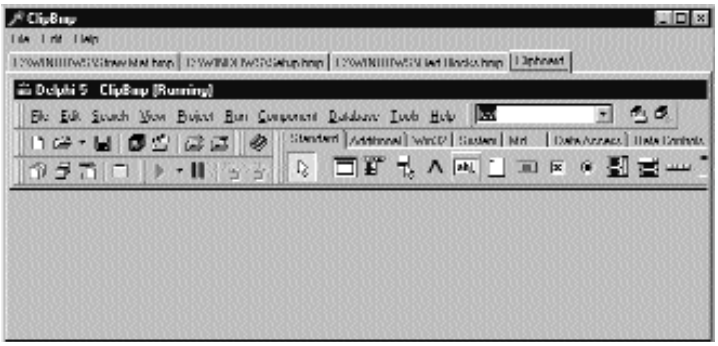
Ove Windows formate možete koristiti bez nekih problema, mada Delphi nema specifičnu podršku za dobijanje ovakvih tipova podataka. U ovom primeru smo koristili dva metoda klase `TMemo` da bismo obavili operacije sa Clipboardom, ali smo isti efekat mogli da postignemo nekim karakteristikama koje se odnose na tekst klase TClipboard. Na primer, možemo koristiti svojstvo `AsText` (koristi se za kopiranje i premeštanje stringova) i metode `SetTextBuf` i `GetTextBuf` (koriste se za rukovanje stringovima `PChar`). Klasa `TClipboard` sadrži specifičnu podršku samo za tekst. Kada želite da radite sa drugim elementima, potrebno je da upotrebite metod `Assign` ili da radite sa hendlovima.

Kopiranje i premeštanje bitmapa

Najčešća tehnika za kopiranje ili premeštanje bitmapa u Delphiju je upotreba metoda `Assign` klasa `TClipboard` i `TBitmap`. Kao nešto napredniji primer upotrebe Clipboarda, načinio sam novu verziju primera PrintBmp koji je ranije pokazan, i nazvao sam novi primer `ClipBmp`. Ovaj program može da prikaže bitmape iz odabranog fajla ili sa Clipboarda ukoliko je dostupan format. Struktura formulara je uvek ista, kada TabControl prekriva ceo formular, a komponenta Image je unutra. Meni je nešto komplikovaniji jer sada sadrži komande za meni Edit.

Kada odaberete komandu Edit➥Paste primera `ClipBmp`, nova kartica nazvana Clipboard se dodaje skupu kartica (izuzev ukoliko već ne postoji), kao što možete videti na slici 19.16. Zatim se koristi broj nove kartice da bi se promenila aktivna kartica:

```
procedure TForm1.Paste1Click(Sender: TObject);
var
  TabNum: Integer;
begin
  {try to locate the page}
  TabNum := TabControl1.Tabs.IndexOf ('Clipboard');
  if TabNum < 0 then
    {create a new page for the Clipboard}
    TabNum := TabControl1.Tabs.Add ('Clipboard');
  {go to the Clipboard page and force repaint}
  TabControl1.TabIndex := TabNum;
  TabControl1Change (Self);
end;
```



SLIKA 19.16 *Strana Clipboard skupa kartica primera ClipBmp prikazuje trenutni sadržaj Clipboarda ukoliko je to bitmapa (u ovom slučaju bitmapa Delphijevog glavnog prozora)*

Na kraju metoda `Paste1Click` program poziva `TabControl1Change`, obradu događaja koja je pridružena selekciji nove kartice, koja može učitati bitmapu iz aktuelnog fajla ili sa Clipboarda:

```
procedure TForm1.TabControl1Change(Sender: TObject);
var
  TabText: string;
begin
  Image1.Visible := True;
  TabText := TabControl1.Tabs [TabControl1.TabIndex];
```

```
  if TabText <> 'Clipboard' then
    {load the file indicated in the tab}
    Image1.Picture.LoadFromFile (TabText)
  else if Clipboard.HasFormat (cf_Bitmap) then
    {if the tab is 'Clipboard' and a bitmap
    is available in the Clipboard}
    Image1.Picture.Assign (Clipboard)
  else
  begin
    {else remove the Clipboard tab}
    TabControl1.Tabs.Delete (TabControl1.TabIndex);
    if TabControl1.Tabs.Count = 0 then
      Image1.Visible := False;
  end;
end;
```

Primetićete da — ako svojstvo Picture komponente Image još nije inicijalizovano — morate da kreirate bitmapu pre pozivanja metoda `Assign`. Ukoliko zaboravite da kreirate novu bitmapu i nijedna grafika nije pridružena slici, operacija `Assign` neće uspeti (i biće pozvan izuzetak). To je zato što metod `Assign` nije konstruktor; to je metod objekta i ukoliko objekat nije kreiran, javiće se izuzetak narušavanja pristupa (access violation) kada pokušate da pozovete jedan od metoda objekta.

NAPOMENA

Metod `Assign` ne čini kopiju bitmape. Njegov efekat je takav da omogućva da se dva objekta `TBitmap` referišu na istu bitmapu u memoriji i na isti hendl bitmape. ■

Ovaj program premešta bitmapu sa Clipboarda svaki put kada promenite karticu. Program čuva samo po jednu sliku i ne sadrži mogućnost čuvanja bitmape Clipboarda. Ipak, ukoliko se sadržaj Clipboarda promeni i ukoliko više nije dostupan format, kartica Clipboard se automatski uklanja (kao što možete videti iz prethodnog listinga). Ukoliko nema više kartica, komponenta Image je sakrivena.

Slika takođe može biti uklonjena upotrebom jedne od dve komande menija: Cut ili Delete. Cut uklanja karticu pošto se načini kopija slike koja se nalazi na Clipboardu. U praksi, metod `Cut1Click` ne čini ništa sem pozivanja metoda `Copy1Click` i `Delete1Click`. Metod Copy1Click je odgovoran za kopiranje slike na Clipboard, a `Delete1Click` jednostavno uklanja trenutnu karticu. Evo koda ovih metoda:

```
procedure TForm1.Copy1Click(Sender: TObject);
begin
  Clipboard.Assign (Image1.Picture.Graphic);
end;
procedure TForm1.Delete1Click(Sender: TObject);
begin
  with TabControl1 do
  begin
    if TabIndex >= 0 then
      Tabs.Delete (TabIndex);
    if Tabs.Count = 0 then
      Image1.Visible := False;
  end;
end;
```

Čuvanje statusa: INI i Registry

Ukoliko želite da sačuvate informacije o statusu aplikacije da biste aplikaciju restaurirali kada se pokrene sledeći put, možete upotrebiti eksplicitnu podršku koju Windows obezbeđuje za čuvanje ovakvog tipa informacija. U prethodnim verzijama Windowsa standardni način je bio kreiranje inicijalizacionog fajla (INI). Kod Windowsa 95/98 i Windowsa NT još uvek možete da koristite INI fajlove, ali Microsoft umesto toga preporučuje upotrebu sistemskog Registryja. U ovom odeljku ćemo razmotriti oba metoda za čuvanje informacija o statusu.

Upotreba Windows INI fajlova

Delphi obezbeđuje klasu koju možete upotrebiti za manipulisanje INI fajlovima, klasu `TIniFile`. Kada ste kreirali objekat ove klase i povezali ga sa fajlom, možete čitati informacije fajla ili zapisivati informacije u fajl. Da biste kreirali objekat, potrebno je da pozovete konstruktor, kome prosleđujete naziv fajla, kao u narednom kodu:

```
var
  IniFile: TIniFile;
begin
  IniFile := TIniFile.Create ('inione.ini');
```

Postoje dva mesta koja možete da odaberete za čuvanje INI fajla. Kod koji je izlistan će sačuvati fajl u Windows direktorijumu (izuzev ukoliko fajl IniOne.ini već ne postoji u direktorijumu aplikacije). Da biste bili bezbedni, bolje je navesti celu putanju u konstruktoru `TIniFile.Create`. Lako možemo da izdvojimo putanju iz naziva programa, kao što sam ja to učinio u primeru IniOne.

Format INI fajlova zahteva neka objašnjenja. Ovi fajlovi su podeljeni u odeljke, od kojih je svaki naznačen nazivom koji se nalazi unutar uglastih zagrada. Svaki odeljak može sadržati veliki broj elemenata tri moguća tipa: stringovi, celi brojevi i Boolean. Ukoliko niste upoznati sa strukturom INI fajla, trebalo bi da pogledate jedan INI fajl, koristeći bilo koji editor teksta, recimo, Windows Notepad.

Klasa `TIniFile` sadrži tri Read metoda, po jedan za svaki tip podataka: `ReadBool`, `ReadInteger` i `ReadString`. Postoje i tri odgovarajuća metoda za zapisivanje podataka: `WriteBool`, `WriteInteger` i `WriteString`. Ostali metodi Vam omogućavaju da pročitate ili uklonite ceo odeljak. Kod Read metoda možete naznačiti unapred određenu vrednost koja će se koristiti ukoliko odgovarajući element ne postoji u INI fajlu.

Naš primer, nazvan IniOne, koristi INI fajl za čuvanje pozicije, veličine i statusa (normalan, maskimiziran ili minimiziran) glavnog formulara. Jedini pravi problem u ovom primeru je to da se vrednost svojstva stanja ne ažurira uvek pravilno VCL-om, te je potrebno da uvedemo dodatni test da biste potvrdili da je formular minimizovan. Glavni formular primera IniOne je samo prazan formular bez ikakvih komponenata. Program obrađuje dva događaja: `OnCreate`, za kreiranje ili otvaranje INI fajla i za čitanje inicijalnih vrednosti, i `OnClose`, za čuvanje statusa posle potvrde korisnika. Evo koda prvog metoda, metoda FormClose, kojim se čuvaju podaci koji će biti pročitani sledeći put kada se pokrene program:

```
procedure TForm1.FormClose(Sender: TObject;
  var Action: TCloseAction);
```

```
var
  Status: Integer;
begin
  if MessageDlg ('Save the current status of the form?',
    mtConfirmation, [mbYes, mbNo], 0) = IdYes then
  begin
    case WindowState of
      wsNormal: begin
        {save position and size, only if the state is normal}
        Inifile.WriteInteger ('MainForm', 'Top', Top);
        IniFile.WriteInteger ('MainForm', 'Left', Left);
        IniFile.WriteInteger ('MainForm', 'Width', Width);
        IniFile.WriteInteger ('MainForm', 'Height', Height);
        Status := 1;
      end;
      wsMinimized: Status := 2;
        {useless: this value is never set by VCL for the main form!}
      wsMaximized: Status := 3;
    end;
    {check if the window is minimized, that is,
    if the form is hidden and not active}
    if not Active then
      Status := 2;
    {write status information}
    IniFile.WriteInteger ('MainForm', 'Status', Status);
  end;
  {in any case destroy the IniFile object}
  IniFile.Free;
end;
```

UPOZORENJE

Glavni formular VCL aplkacije se nikada ne smanjuje — kada glavni formular primi poruku minimizacije, ona se prosleđuje prozoru aplikacije koji smanjuje celu aplikaciju. `WindowState` kao rezultat daje vrednost `swMinimize` za sekundarne formulare aplikacije koji se minimiziraju, ali ne i glavni formular. Da biste znali da li je aplikacija minimizovana, možete proveriti da li je glavni formular aktivan, kao u prethodnom kodu. ■

Drugi metod je `FormCreate`, koji jednostavno čita sačuvane podatke i restaurira prethodnu situaciju. Obratite pažnju na kod inicijalizacije, koji traži INI fajl u direktorijumu programa, uzimajući njegov naziv i menjajući mu ekstenziju. Evo kompletnog koda:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  Status: Integer;
begin
  IniFile := TIniFile.Create (ChangeFileExt (
    Application.ExeName, '.ini'));
  {try to read a value and test if it exists}
  Status := IniFile.ReadInteger ('MainForm', 'Status', 0);
  if Status <> 0 then
  begin
    {read position and size using current values as default}
    Top := IniFile.ReadInteger ('MainForm', 'Top', Top);
    Left := IniFile.ReadInteger ('MainForm', 'Left', Left);
```

```
    Width := IniFile.ReadInteger ('MainForm', 'Width', Width);
    Height := IniFile.ReadInteger ('MainForm', 'Height', Height);
    {set the minimized or maximized status}
    case Status of
      // 1: WindowState := wsNormal;
        // this is already the default
      2: WindowState := wsMinimized;
      3: WindowState := wsMaximized;
    end;
  end;
end;
```

Ovaj kod koristi polje nazvano `IniFile`, tipa `TIniFile`, koje je dodato privatnom odeljku klase `TForm1`. Ja nisam obezbedio sliku koja prikazuje izlaz programa, jer nije od naročite pomoći da Vam prikažem prazan formular ili ikonu. Umesto toga, trebalo bi pokrenete program nekoliko puta i da istražite njegovo ponašanje, menjajući svaki put veličinu i poziciju programa. Ono što ću Vam prikazati je primer INI fajla koji generiše program:

```
[MainForm]
Top=359
Left=567
Width=217
Height=201
Status=1
```

SAVET

Delphi veoma često koristi INI fajlove, ali su oni prerušeni različitim nazivima. Na primer, fajlovi radne površine (`.dsk`) i opcija (`.dof`) su struktuirani kao INI fajlovi. ■

Upotreba Registryja

Sada možemo napisati sličan program koristeći sistemski Registry umesto običnih INI fajlova. Pre nego što napišemo program, želim da ukratko razmotrim ulogu i strukturu Registryja. U osnovi, Registry je hijerarhijska baza podataka informacija o kompjuteru, konfiguraciji softvera i opcijama koje je korisnik odabrao. Windows sadrži skup API funkcija za komunikaciju sa Registryjem; u osnovi otvarate ključ (ili folder), a zatim radite sa potključevima (ili potfolederima) i njihovim vrednostima (ili elementima), ali morate znati strukturu i detalje o Registryju. Windows 95/98 Registry je zasnovan na šest ključeva najvišeg nivoa.

Pogledajte odgovarajuću Microsoftovu dokumentaciju (recimo, Resource Kit) da biste saznali detalje o organizaciji Registryja i informacijama gde da dodate sopstvene ključeve. Važnost Registryja se ne sme potceniti. Registry čuva kritične informacije o konfiguraciji hardvera sistema, podešavanjima Control Panela i OLE serverima, a sadrži i statistiku o mašini. Da biste prostudirali strukturu Registryja i produžili trenutne vrednosti ključeva, možete pozvati program RegEdit.

UPOZORENJE

Program RegEdit možete upotrebiti i za izmenu vrednosti Registryja, ali bi bilo bolje da izbegnete bilo kakve izmene izuzev ukoliko niste sigurni da znate šta radite. ■

Delphi u osnovi obezbeđuje dva pristupa za upotrebu Registryja i svaki podržava VCL klasom: klasom `TRegistry` i klasom TRegIniFile. Prva klasa obezbeđuje generičku enkapsulaciju Registry API-ja, dok druga obezbeđuje interfejs (metode i svojstva) klase `TiniFile`, ali podatke čuva u Registryju umesto da koristi fajlove. Ova klasa je prirodan izbor za verziju Registry našeg prethodnog primera programa; njenom upotrebom nećemo morati da načinimo previše izmena u izvornom kodu (prava prednost kad god već imate kod koji je zasnovan na INI fajlovima). Evo tri izmene koje morate načiniti u programu `IniOne`:

1. Upotrebite `TRegIniFile` umesto `TIniFile` kao klase objekta `IniFile`.
2. Kreirajte novi objekat `TRegIniFile` umesto objekta `TIniFile` u metodu `FormCreate`:

   ```
   IniFile := TRegIniFile.Create ('IniOne.ini');
   ```

3. U iskazu `uses` u interfejs odeljku jedinice zamenite jedinicu IniFiles jedinicom Registry.

Lako, zar ne? Ovim jednostavnim izmenama ja sam izradio primer Registr, koji ima identične mogućnosti kao i prethodni primer, ali podatke čuva u Registryju umesto u INI fajlu. Zapravo, kada koristite klasu `TRegIniFile`, Delphi dodaje novi potključ sa nazivom INI fajla pod ključem `HKEY_CURRENT_USER`. Umesto da svoje elemente dodajete direktno pod ovaj ključ, trebalo bi da ih smestite pod potključ `Software` i da možda dodate jedan ili više nivoa za svoju softversku kompaniju. Sledi kod primera Registr:

```
IniFile := TRegIniFile.Create (
  'Software\Mastering Delphi\Registr');
```

Efekat ovog koda možete videti ukoliko istražite Registry aplikacijom RegEdit, kao što je prikazano na slici 19.17.

SLIKA 19.17 *Upotrebom programa Registr možete dodati nove elemente bazi podataka registracija, kao što možete videti ukoliko pregledate Registry upotrebom programa RegEdit*

Klasa `TRegIniFile` je zapravo potklasa klase TRegistry, koja sadrži veliki broj metoda koji su slični funkcijama Registry API-ja. Ove funkcije nisu lake za upotrebu, te Vam predlažem da se u većini slučajeva držite jednostavnije klase TRegIniFile. Da biste upotrebili klasu TRegistry, potrebno je da prvo otvorite ključ i da zatim pristupite podacima ključa, uključujući njegove vrednosti i potključeve.

Da bih Vam pokazao osnovne mogućnosti klase `TRegistry`, ja sam izradio veoma jednostavnu aplikaciju za prikaz Registryja. Ovaj program može da prikaže strukturu Registryja i vrednosti ključeva i elemenata, ali ne prikazuje stvarne podatke koji su povezani sa ključevima i elementima. Ovaj program sadrži samo podskup mogućnosti aplikacije RegEdit, ali ja smatram da je ovo ipak interesantan primer.

Program RegView je zasnovan na formularu sa dva combo polja i dve liste. Kada se aplikacija pokrene, kreira se objekat `TRegistry`:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Reg := TRegistry.Create;
  Reg.OpenKey ('\', False);
  UpdateAll;
  // select the current root
  ComboKey.ItemIndex := 1;
  ComboLast.Items.Add ('\');
  ComboLast.ItemIndex := 0;
end;
```

Ovaj kod otvara unapred određeni koreni ključ (koji je naznačen karakterom \), a zatim ažurira korisnički interfejs. Na kraju selektuje unapred određeni koreni ključ u prvom combo polju (o ovome će malo kasnije biti više reči) i dodaje trenutni element `ComboLast` combo polju. Metod `UpdateAll` jednostavno kopira trenutnu putanju u naslov formulara i popunjava dve liste potključevima i vrednostima trenutnog ključa (kao što možete videti na slici 19.18):

```
procedure TForm1.UpdateAll;
begin
  Caption := Reg.CurrentPath;
  if Reg.HasSubKeys then
    Reg.GetKeyNames(ListSub.Items)
  else
    ListSub.Clear;
  Reg.GetValueNames(ListValues.Items);
end;
```

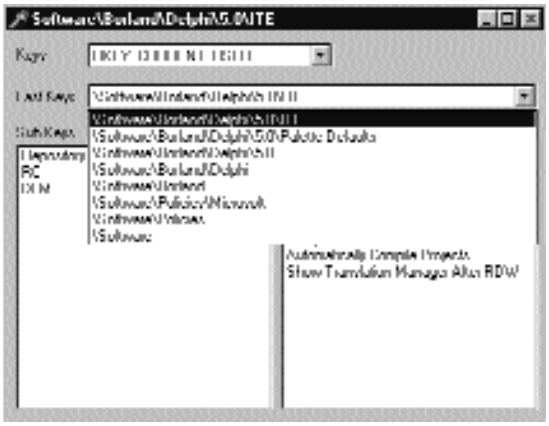
Kada iz prve liste odaberete element (`ListSub`), program prelazi na odabrani potključ. Da biste ovo postigli, jednostavno napišite naredni kod:

```
procedure TForm1.ListSubClick(Sender: TObject);
var
  NewKey: string;
begin
  NewKey := ListSub.Items [ListSub.ItemIndex];
  Reg.OpenKey (NewKey, False);
  UpdateAll;
end;
```

Ovo je sasvim dovoljno za kretanje kroz celo drvo. Dva combo polja dodaju još neke mogućnosti programu. Prvo polje prikazuje moguće korene ključeva za Windows 95 i 98. Kada se selekcija promeni, odgovarajući sadržaj se selektuje kao koreni ključ objekta `TRegistry`:

```
procedure TForm1.ComboKeyChange(Sender: TObject);
begin
  case ComboKey.ItemIndex of
    0: Reg.RootKey := HKEY_CLASSES_ROOT;
    1: Reg.RootKey := HKEY_CURRENT_USER;
    2: Reg.RootKey := HKEY_LOCAL_MACHINE;
    3: Reg.RootKey := HKEY_USERS;
    4: Reg.RootKey := HKEY_CURRENT_CONFIG;
    5: Reg.RootKey := HKEY_DYN_DATA;
```

```
  end;
  Reg.OpenKey ('\', False);
  UpdateAll;
  ComboLast.Items.Clear;
end;
```



SLIKA 19.18 *Izlaz primera RegView, koji prikazuje ključeve i vrednosti Registryja (odeljak sa Delphijevim podešavanjima). Combo polje prikazuje spisak Registry ključeva koji su korišćeni u poslednje vreme*

Posle određivanja korenog ključa, program otvara njegov koreni element, ažurira korisnički interfejs i uklanja elemente iz drugog combo polja, Last Keys. Ovo polje čuva dnevnik, listu prethodnih izbora i može se upotrebiti za prolazak kroz drvo, a da ne morate svaki put ponovo krenuti od korenog ključa. Evo njegovog koda:

```
procedure TForm1.ComboLastChange(Sender: TObject);
begin
  Reg.OpenKey (ComboLast.Text, False);
  UpdateAll;
end;
```

Ovaj kod je jednostavan, ali kod koji je potreban za ažuriranje liste elemenata ovog combo polja je prilično komplikovan. Pored provere da li putanja već postoji, mi moramo dodati obrnutu kosu crtu (\) ispred bilo koje putanje koja je nema na početku. Problem je to što kod koji je izlistan ne određuje pravilno svojstvo `CurrentPath` objekta `Reg`, mada proizvodi korektan efekat. Da bismo u budućnosti načinili izbor putanje (kada je dodata listi), moramo korigovati listu. Uzimajući u obzir šta sve treba učiniti, evo finalne verzije metoda `ListSubClick`:

```
procedure TForm1.ListSubClick(Sender: TObject);
var
  NewKey, Path: string;
  nItem: Integer;
begin
  // get the selection
  NewKey := ListSub.Items [ListSub.ItemIndex];
  Reg.OpenKey (NewKey, False);
  // save the current path (eventually adding a \)
```

```
  // only if the it is not already listed
  Path := Reg.CurrentPath;
  if Path < '\' then
    Path := '\' + Path;
  nItem := ComboLast.Items.IndexOf (Path);
  if nItem < 0 then
  begin
    ComboLast.Items.Insert (0, Path);
    ComboLast.ItemIndex := 0;
  end
  else
    ComboLast.ItemIndex := nItem;
  UpdateAll;
end;
```

Pristupanje svojstvima po nazivu

Kao što znate, Object Inspector prikazuje spisak published svojstava objekta, čak i za komponente koje ste Vi napisali. Da bi ovo postigao, Object Inspector se oslanja na RTTI informaciju koja se generiše za published svojstva. Upotrebom nekih naprednih tehnika, aplikacija može pročitati listu published svojstava objekta i može ih upotrebiti.

Mada ova mogućnost nije poznata, u Delphiju je moguće pristupiti svojstima prema nazivu jednostavnom upotrebom stringa sa nazivom svojstva, a zatim dobiti vrednost svojstva. Pristup RTTI informacijama je obezbeđen preko grupe nedokumentovanih podrutina, koje su deo jedinice TypInfo.

UPOZORENJE

Razlog što ove rutine nisu dokumentovane je to što Borland želi da ih u budućim verzijama Delphija slobodno menja. U prošlosti (od Delphija 1 do Delphija 4) ove funkcije su se veoma malo izmenile. Delphi 5 TypInfo obezbeđuje mnogo toga lepog, a ovo je slučaj nekoliko nekompatibilnosti. Ukoliko bilo šta koristite iz TypInfoa, imajte na umu da je možda potrebno da ažurirate svoj kod za buduće verzije Delphija. ■

Umesto da istražimo celu jedinicu TypInfo, mi ćemo ovde pogledati samo minimalan kod koji je neophodan za pristupanje svojstvima prema nazivu. Pre Delphija 5 je bilo neophodno da koristite funkciju `GetPropInfo` da biste dobili pokazivač na neke informacije o internim svojstvima i da zatim primenite jednu od funkcija za pristupanje, kao što je GetStrProp, na ovaj pokazivač. Takođe je trebalo da proverite postojanje i tip svojstva.

Delphi 5 predstavlja novi skup TypInfo rutina, uključujući zgodnu rutinu `GetPropValue`, koja kao rezultat daje Variant sa vrednošću svojstva, ili `NULL` ukoliko svojstvo ne postoji. Jednostavno ovoj funkciji prosledite objekat i string sa nazivom svojstva. Naredni opcioni parametar Vam omogućava da odaberete format povratnih vrednosti svojstava tipa skupa.

Na primer, možemo načiniti poziv

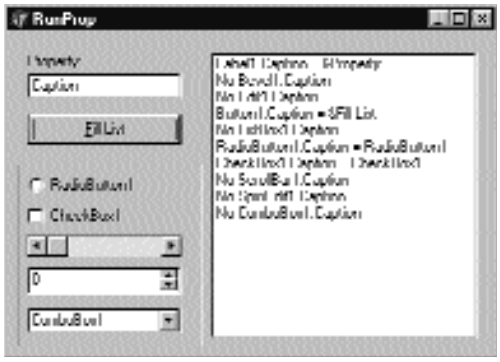
```
ShowMessgae (GetPropValue (Button1, 'Caption'));
```

Ovaj poziv ima isti efekat kao i pozivanje funkcije `ShowMessage` kada joj se kao parametar prosleđuje `Button1.Caption`. Jedina prava razlika je u tome da je ova verzija koda mnogo sporija, jer kompajler generalno rešava normalan pristup svojstvima na efikasniji način. Prednost pristupa u vreme izvršavanja je u tome da ga možete načiniti veoma fleksibilnim, kao u narednom primeru RunProp.

Ovaj program u listi prikazuje vrednost svojstva bilo kog tipa za svaku komponentu formulara. Naziv svojstva koje tražimo se daje u polju za izmene. Ovo program čini veoma fleksibilnim. Pored polja za izmene i liste, formular sadrži i kontrolu za generisanje izlaza i nekoliko komponenata koje su dodate samo radi testiranja njihovih svojstava. Kada kliknete kontrolu, izvršava se sledeći kod:

```
uses
  TypInfo;
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
  Value: Variant;
begin
  ListBox1.Clear;
  for I := 0 to ComponentCount -1 do
  begin
    Value := GetPropValue (Components[I], Edit1.Text);
    if Value <> NULL then
      ListBox1.Items.Add (Components[I].Name + '.' +
        Edit1.Text + ' = ' + string (Value))
    else
      ListBox1.Items.Add ('No ' + Components[I].Name + '.' +
        Edit1.Text);
  end;
end;
```

Efekat upotrebe kontrole Fill List, ako se koristi unapred određena vrednost *Caption* u polju za izmene, možete videti na slici 19.19. Možete isprobati bilo koji drugi naziv svojstva. Brojevi će biti konvertovani u stringove variant konverzijom. Objekti (kao što je vrednost svojstva `Font`) će biti prikazani kao memorijske adrese.



SLIKA 19.19 *Izlaz primera RunProp, koji pristupa svojstvima u vreme izvršavanja prema nazivu*

Nemojte stalno koristiti jedinicu TypInfo umesto drugih tehnika za pristupanje svojstvima. Prvo upotrebite pristup svojstvima osnovne klase, ili upotrebite bezbednu konverziju `as` kada je potrebno, a RTTI pristup svojstvima ostavite kao poslednju mogućnost. Upotreba tehnika TypInfo čini Vaš kod sporijim, složenijim i podložnijim greškama; ova tehnika preskače proveru tipova za vreme kompajliranja i čini kod manje pokretnim za buduće verzije Delphija.

Izrada online helpa

Izuzev ukoliko su Vaše aplikacije veoma jednostavne, obično ćete želeti da dodate neki oblik pomoći da biste odgovorili na pitanja koja se javljaju kada korisnici rade sa Vašim softverom. Windows obezbeđuje podršku za pomoć (help) na nivou operativnog sistema, ali Vi još uvek morate uraditi priličan deo posla da biste obezbedili help za svoje aplikacije. Da biste pojednostavili ovaj zadatak, možete da upotrebite nekoliko proizvoda nezavisnih programera, kao što su ForeHelp, RoboHelp i nekoliko drugih. Ipak, online help možete da izradite upotrebom alata koje dobijate uz Delphi i uz pomoć programa za obradu teksta koji mogu da generišu tekst fajlove RTF formata. Mi ćemo razmotriti osnovne korake koji su potrebni za kreiranje helpa na ovaj način, ali imajte na umu da alati nezavisnih programera generišu iste fajlove i čine iste stvari. Ovi programi takođe mogu da automatizuju mnoge elemente procesa koji se ponavljaju i koji su podložni greškama.

Postoje tri glavna fajla koja je potrebno da generišete pre nego što možete da kreirate online help (HLP) fajl: fajl projekta, fajl sadržaja i sam fajl sa tekstom helpa. Tek kada napravite ova tri fajla, možete upotrebiti Microsoftov Help Compiler da biste sve spojili u HLP fajl. Microsoft Help Workshop, koji dobijate uz Delphi, može da se koristi za kreiranje fajlova projekta i fajlova sadržaja ukoliko ne nameravate da koristite alat nezavisnih programera. Upotrebom Help Workshopa i MS Worda (ili nekog drugog programa za obradu teksta koji može da generiše RTF fajlove) spremni ste da kreirate fajlove za online help. Podelimo ovaj proces u četiri etape: kreiranje fajla/fajlova sa tekstom helpa, kreiranje fajla Contents, kreiranje fajla Project i mapiranje/povezivanje elemenata helpa sa elementima Vaše Delphi aplikacije.

Kao što sam ranije pomenuo, da biste kreirali fajlove sa tekstom helpa, potreban Vam je program za obradu teksta koji može da generiše tekst fajlove RTF formata, kakvi su WordPerfect ili MS Word. Kada počnete da unosite tekst, možete da ga smestite u više fajlova (help menija u `Menu.rtf`, help okvira za dijalog u `Dialog.rtf` i tako dalje), ili da sav tekst smestite u jedan fajl. Slede moji saveti za korake koje treba izvršiti prilikom kreiranja teksta helpa:

- Kreirajte help stranu za svaki formular i okvir za dijalog. Svakoj strani dajte naslov prema nazivu formulara ili okvira za dijalog.
- Kreirajte help stranu za svaki element menija koji ne prikazuje okvir za dijalog. Ovim stranama dajte naslov prema nazivu menija, za kojim sledi naziv elementa menija, a odvojte nazive uspravnom crtom (|), kao Edit | Copy.
- Kreirajte iskačuće (pop up) help strane za termine koji mogu biti nepoznati korisniku. Ovim stranama ne morate da date naslov jer ćete tekst prikazati korisniku u prozoru koji će biti pored nepoznatog termina.

- Kreirajte strane helpa koje sadrže linkove na teme koje imaju veze sa temom koja se prikazuje, recimo, kao opcije komandne linije.
- Kreirajte help strane za uobičajene zadatke i kreirajte linkove na strane za elemente menija, formulare i okvire za dijalog koji su potrebni za obavljanje nekog zadatka.
- Kreirajte po potrebi strane sa instrukcijama.
- Kreirajte help stranu koja sumira glavne karakteristike i upotrebu proizvoda.
- Ukoliko je proizvod nova verzija postojećeg proizvoda, kreirajte help stranu koja sumira nove mogućnosti.

Razmotrimo sada kako se formatira osnovna help strana. Većina help strana sadrži sledeće elemente (pogledajte `HCW.HLP` za detaljnije informacije o formatiranju help strana):

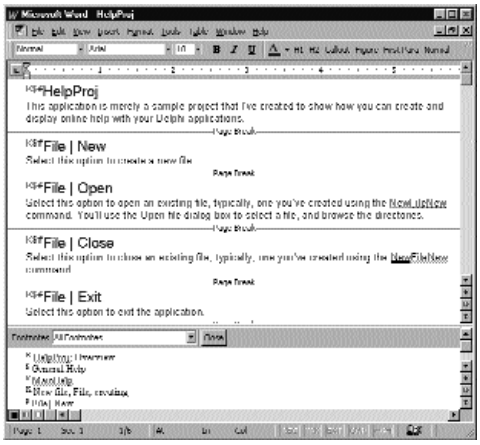
- Naslovni tekst teme, obično ispisan masnim (bold) slovima, ponekad i drugačijom bojom od tela teksta, i obično veličine 14 pointa ili veći.
- Kontekst string za stranu, unosi se kao # fusnota naslova teme. Ovo je string koji help sistem koristi za jedinstvenu identifikaciju te help strane. Mora biti jedinstven i ne može sadržati razmake i znake interpunkcije.
- Tekst string za pretraživanje, unosi se kao $ fusnota naslova teme. To je string koji će korisnik videti u okviru za dijalog Search, listi History i meniju Bookmark. Obično se želi da ovaj tekst odgovara tekstu naslova teme.
- Stringovi ključnih reči za temu, unose se kao K fusnota naslova teme. Ovi stringovi će se prikazati u kartici Index. Možete uneti više od jednog stringa ključne reči, ali stringove morate odvojiti tačkom i zarezom. Da biste prikazali različite podnaslove u indeksu pod datim glavnim naslovom, kao prvu reč stringa unesite naslov, a podnaslov kao drugu reč, i odvojte ih zarezom.
- Linkove na druge help strane, unesene kao tekst koji je dva puta podvučen i za kojim odmah sledi kontekst string, a formatiran je kao sakriveni tekst.
- Kraj strane. Sve help strane se moraju završiti na ovaj način.

Uz to, imajte na umu i sledeće kada formatirate svoje help strane:

- Da biste na vrhu strane kreirali oblast koja se ne može skrolovati, upotrebite stil paragrafa Keep With Next za naslov i linkove (kao što je link "Related Topics") koje želite da prikažete u oblasti koja ne može da se skroluje.
- Za veći deo teksta helpa ćete želeti da koristite font Arial, jer se lako čita u većini veličina fonta i standardni je font koji će biti na svakom sistemu.
- Možete načiniti linkove na teme koje se nalaze u drugim HLP fajlovima. Ukoliko znate kontekst stringove u drugim HLP fajlovima, možete se povezati na te teme upotrebom standardnih linkova, ali morate drugačije formatirati kontekst string. Umesto unošenja kontekst stringa kao sakrivenog teksta, potrebno je da unesete naziv HLP fajla, asterisk, a zatim kontekst string (bez razmaka između), sve kao sakriven tekst. Ukoliko nemate pristup kontekst stringovima, potrebno je da

upotrebite makro JumpKeyword, koji je dokumentovan u online helpu Help Workshopa.

U ovom trenutku bi trebalo da ste kreirali sav tekst i formatirali ga po stranama. Na slici 19.20 je prikazan primer help tekst fajla u RTF formatu, sa nekoliko kratkih help strana.



SLIKA 19.20 *Jednostavan help tekst fajl u RTF formatu*

Razmotrimo sada kako ćete generisati Help Contents fajl. Contents i Project fajlovi nisu binarni fajlovi — to su tekst fajlovi. Mnogo je lakše koristiti Help Workshop za formatiranje i o državanje ovih fajlova iako su tekst fajlovi.

U Help Workshopu ćete kreirati novi Contents fajl, koji zahteva da navedete naziv HLP fajla, naziv tipa glavnog help prozora i unapred određeni naslov koji želite da se prikaže na vrhu help prozora. Zatim, kreirate zaglavlja i teme koje će se prikazati u kartici Contents prozora Contents/Index/Search. Za svaku temu ćete navesti prvu temu teksta, a zatim kontekst string help strane koju želite da pridružite temi. Primer editovanja Contents fajla možete videti na slici 19.21.

SLIKA 19.21 *Contents fajl, onako kako se prikazuje u Help Workshopu*

Kada ste spremni da kreirate Project fajl, možete ga kreirati iz Help Workshpa. U Project fajlu navodite sledeće kritične informacije:

- nazive RTF fajlova koji sadrže tekst za help strane,
- naziv Contents fajla,
- naziv rezultujućeg HLP fajla,
- tehnike kompresovanja koje želite da se upotrebe, ukoliko želite,
- mapu odgovarajućih kontekst stringova i kontekst ID,
- veličinu, poziciju, boju i izgled help prozora.

Razmotrimo sada poslednje dve informacije nešto detaljnije. Kontekst ID broj mape je neophodan za prikazivanje help informacije za elemente menija, formulare ili komponente. Kada kreirate kontekst string i ovaj broj mape, želite da koristite standardne tehnike za dodeljivanje ovih brojeva. Savetujem Vam da koristite sledeći sistem numerisanja:

- Mapirajte kontekst string za help stranu rezimea ili pregleda na ID 0 i 1.
- Mapirajte kontekst string za help stranu sa novim mogućnostima (ukoliko ih ima) na ID 2.
- Brojeve između 3 i 99 upotrebite za ID brojeve za iskačuće strane, strane sa povezanim temama, strane uobičajenih zadatka i strane sa uputstvima.
- Brojeve između 100 i 199 upotrebite kao ID brojeve za prvi meni, njegove elemente menija i rezultujuće okvire za dijalog. Na primer, ID 100 bi opisao meni i obezbedio linkove za help svakog elementa menija. ID 101 bi opisao prvi element menija, 102 drugi element i tako dalje. Zatim biste upotrebili brojeve između 130 i 199 za opisivanje okvira za dijalog koji se prikazuju za elemente prvog menija.
- Upotrebite brojeve između 200 i 299 za drugi meni, njegove elemente menija i rezultujuće okvire za dijalog. Označite brojevima preostale menije, elemente menija i okvire za dijalog prema ovoj šemi.

Upotrebom ove tehnike možete brzo da dodelite ID brojeve kontekst stringovima i možete lakše da organizujete help strane.

Kao što ste možda primetili, različite aplikacije prikazuju online help na različitim početnim pozicijama. Možda i Vi želite da ovo učinite da biste osigurali da online help ne zaklanja važan formular ili prozor, ili želite da help prozor bude što je moguće veći. Pored određivanja veličine prozora i njegove pozicije u Project fajlu, takođe možete odrediti boju pozadine za tekst (kao i za oblasti koje se ne skroluju, ukoliko ih kreirate). Da biste primenili ove vrednosti na help prozor, kreirajte novi stil prozora koji ćete nazvati "Main" i tu smestite izmene.

Kada ste kreirali Project fajl i kada ste naveli odgovarajuće parametre, možete da kompajlirate i testirate HLP fajl. Help Workshop prikazuje poruke o greškama i detaljan izveštaj o ovom procesu, informišući Vas o broju tema, linkova, ključnih reči i bitmapa koje sadrži Vaš HLP fajl. Takođe možete direktno pokrenuti WinHelp iz Help Workshpa, koji Vam omogućava da testirate HLP

fajl pre nego što pređete na poslednji korak, poveizvanje tema sa određenim elementima u Vašoj Delphi aplikaciji.

Povezivanje Vaše Delphi aplikacije sa HLP fajlom je, zapravo, veoma jednostavno; sastoji se od samo nekoliko koraka:

- Navedite naziv HLP fajla u okviru za dijalog Project Options aplikacije.
- Odredite vrednost 1 za svojstvo `HelpContext` glavnog formulara.
- Podesite svojstvo `HelpContext` ostalih komponenata na odgovarajuće vrednosti, na osnovu ID mapiranja koje ste načinili u Project fajlu.
- Ukoliko niste kreirali help stranu za neku komponentu, meni ili element menija, ostavite vrednost 0 za svojstvo `HelpContext`, jer je ovo vrednost koja je mapirana na rezime.
- Upotrebite standardne akcije koje se odnose na Help (`THelpContents`, `THelpTopic Search` i `THelpOnHelp`) da biste implementirali meni Help za nekoliko sekundi. Ove akcije su novina u Delphiju 5.

To je sve! Kada ste testirali aplikaciju, napisali online help i testirali help sistem (i zatim ponovo testirali aplikaciju nekoliko puta), spremni ste da kreirate instalacioni program.

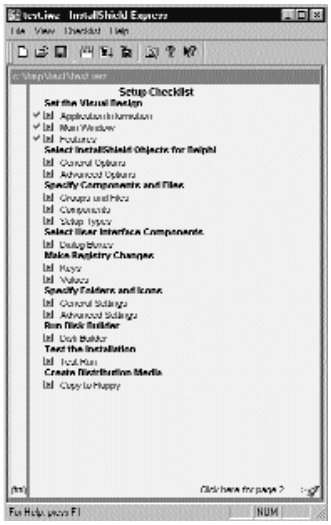
InstallShield Express

Gotovo svaka komercijalna aplikacija koristi neku vrstu setup programa za manipulisanje instalacijom i konfiguracijom aplikacije pre prve upotrebe. Među mnogim dobrim pomoćnim programima na tržištu za kreiranje setup programa, jedan od najpopularnijih je InstallShield. Delphi dobijate sa verzijom ovog prgrama koji ima nešto manje mogućnosti, a koji je poznat kao InstallShield Express. Čak i tada, za kompletnu obradu ove teme bi bila potrebna čitava knjiga, a Vi možete prilično lako da kreirate jednostavan setup program upotrebom InstallShielda.

Postoje mnoge prednosti upotrebe alata nezavisnih programera kakav je alat InstallShield za kreiranje Vaših setup programa, a kako se povećava složenost Vaše aplikacije, tako se povećava i vrednost ovakvog tipa alata. Na primer, ukoliko Vaše aplikacije zahtevaju BDE, ili bilo koji od BDE drajvera, Vi jednostavno birate koje od tih komponenata su Vam potrebne, a InstallShield obrađuje sve detalje. Slično tome, ukoliko pišete novu verziju složene aplikacije koja kod širi na nekoliko DLL-ova, InstallShield može da kreira setup program koji detektuje informacije o verziji u postojećoj instalaciji i ažurira samo neophodne fajlove. Treba da instalirate skup fajlova za Windows 95 ili 98, a drugi skup fajlova za Windows NT? Možete i to da učinite. Razmotrimo ukratko kako se koristiti InstallShield Express za kreiranje setup programa.

Kada koristite InstallShield Express za kreiranje novog setup programa, počećete navođenjem tri važna dela informacija: naziv projekta, poddirektorijum gde želite da smestite fajlove projekta i da li želite ili ne da upotrebite instalaciju sa prilagođavanjem. Ove informacije unosite u okvir za dijalog New Project. Tačnije, trebalo bi da obratite pažnju na Include a Custom Setup Type da biste načinili ispravan izbor. Ukoliko ga sada ne selektujete, kasnije nećete moći da dodate instalaciju sa podešavanjem.

Kada ste kreirali novi InstallShield Express projekat, prikazaće se Setup Checklist, koji se vizuelno smešta na beležnicu, kao što je prikazano na slici 19.22. Mada je prirodno navesti ove elemente po redosledu u kojem se pojavljuju, možete ih u opštem slučaju navesti u bilo kom redosledu (izuzev za izradu, testiranje i za uređivanje setup aplikacije, koja se, naravno, mora obaviti poslednja). Svaki odeljak sadrži jedan ili više elemenata liste, od kojih svaki odgovara kartici okvira za dijalog za taj odeljak. Ukratko, razmotrimo svaki od glavnih odeljaka liste i proučimo neke akcije koje treba da obavite.



SLIKA 19.22 *Korisnički interfejs InstallShield Expressa je baziran na metafori beležnice*

Prvi odeljak, Visual Design, sadrži tri elementa, a okvir za dijalog Set the Visual Design će prikazati tri odgovarajuće kartice. U kartici App Info navodite naziv Vaše setup aplikacije, putanju izvršnog fajla i naziv fajla, verziju i naziv Vaše kompanije. U kartici Main Window navodite naslov Vašeg setup programa, bitmapu logotipa i boju pozadine glavnog prozora setup programa. U kartici Features navodite da li InstallShield Express treba da za Vas kreira opciju za deinstaliranje. (Ja Vam savetujem da upotrebite ovu opciju, jer gotovo da ne postoje nedostaci kada je odaberete.)

Kada ste u okvir za dijalog uneli odgovarajuće podatke, kliknite OK da biste sačuvali podatke. Kada se ponovo pojavi prozor Setup Checklist, primetićete oznaku pored svakog elementa u odeljku Visual Design. Ove oznake Vas jednostavno podsećaju na to koje elemente ste uneli.

Drugi odeljak Setup Checklista je nazvan Select InstallShield Objects for Delphi; on sadrži odeljke kao što su General i Advanced. Rezultujući okvir za dijalog ćete koristiti da biste odabrali opcione Delphi komponente ili dodatke koje želite da instalirate. Na primer, ukoliko kreirate setup program za aplikaciju za bazu podataka, bez sumnje ćete instalirati Borland Database Engine (BDE), što može biti komplikovano, čak i za jednostavne aplikacije. U ovom okviru za dijalog ne samo da možete odabrati da li da instalirate BDE ili ne, već takođe možete odabrati koje delove BDE-a želite da instalirate.

Treći odeljak, Specify Components and Files, sadrži tri elementa, što još jednom znači tri odeljka u odgovarajućem okviru za dijalog. U ovom okviru za dijalog možete navesti Groups and Files, Components i Setup Types. Zbog različtiog manipulisanja ovim terminima njihovo značenje ne mora biti očigledno, te ću ih ja ukratko objasniti:

- *File Group* je logički skup fajlova koji setup program mora da instalira u navedeni direktorjium. Na primer, obično želite da sve fajlove online helpa smestite u isti direktorijum. Dok je ovo u opštem slučaju isti direktorijum kao i direktorijum aplikacije, to ne mora biti pravilo. Ove fajlove ćete odvojiti od ostalih fajlova u grupi Program Files, jer postoje mnoge situacije kada ne želite da ih instalirate. Kada ste u nedoumici, kreirajte novu grupu fajlova za jedan ili više fajlova kada ih instalirate kao opcione.
- *Component* je skup od jedne ili više grupa fajlova. Kada su grupe fajlova razdvojene na osnovu odredišnog direktorijuma, komponente ćete razdvojiti na osnovu njihove logičke funkcije. Na primer, ukoliko nećete da instalirate neke opcione online help fajlove, verovatno ne želite da instalirate odgovarajuće fajlove sa uputstvom koje ste pripremili. Kreiranjem komponente koja sadrži grupu help fajlova za zamenu i grupu fajlova sa uputstvom, možete navesti da ne želite da ih instalirate kao skup. Mada su logički grupisane kao komponenta, ukoliko ih instalirate, setup program će ispoštovati odredišne direktorijume koje ste naveli za grupe fajlova.
- *Setup Type* je logička grupa komponenata. Unapred su određena tri standardna tipa setupa za instaliranje komponenata, ali Vi to možete da prilagodite eliminacijom komponente (to jest, skupa grupa fajlova) iz određenog tipa setupa.

Naredni odeljak u prozoru Setup Checklist je nazvan Select User Interface Components. Izborom elementa Dialog Box možete da navedete koje standardne setup okvire za dijalog želite da prikažete korisniku. Evo spiska okvira za dijalog koje možete da odaberete:

WELCOME BITMAP Mada se ne koristi po definiciji, možete prikazati početnu bitmapiranu sliku prilikom pokretanja svog setup programa.

WELCOME MESSAGE Odaberite ovaj okvir za dijalog da biste prikazali standardnu tekstualnu pozrdavnu poruku.

SOFTWARE LICENSE AGREEMENT Odaberite ovu opciju da biste prikazali tekst iz svoje licence.

README INFORMATION Isto kao i licenca, ali ćete ovde prikazati informacije koje treba pročitati.

USER INFORMATION Korisnik unosi ime, adresu i, opciono, serijski broj prizvoda.

CHOOSE DESTINATION INFORMATION Korisnik bira odredišni uređaj i/ili direktorijum.

SETUP TYPE Korisnik selektuje opcije između Typical, Compact ili Custom (opcija Custom će biti dostupna samo ukoliko ste naveli da je InstallShield generiše).

CUSTOM SETUP Ukoliko je odabrano Custom za opciju Setup Type, ovaj okvir za dijalog omogućava korisnicima da odaberu komponente koje žele da instaliraju.

SELECT PROGRAM FOLDER Određuje unapred definisani naziv foldera koji će sadržati aplikaciju i prateće fajlove.

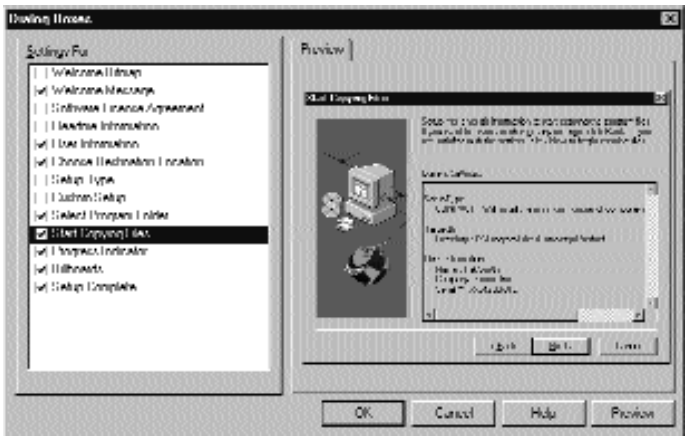
START COPYING FILES Započinje proces instaliranja.

PROGRESS INDICATOR Vizuelno prikazuje korisniku status setup procesa.

BILLBOARDS Upotrebite ovu opciju da biste prikazali jedan ili više BMP fajlova dok korisnik izvršava setup program.

SETUP COMPLETE Ovde ćete odabrati prirodu poruke koja se prikazuje kada je korisnik obavio instaliranje programa.

Možete odabrati da prikažete sve ove okvire za dijalog, ni jedan od njih, ili bilo koju kombinaciju koju želite. Setup program će ih prikazati korisniku u bilo kom redosledu u kojem se pojavljuju u okviru za dijalog. Na slici 19.23 je prikazan okvir za dijalog Select User Interface Components.



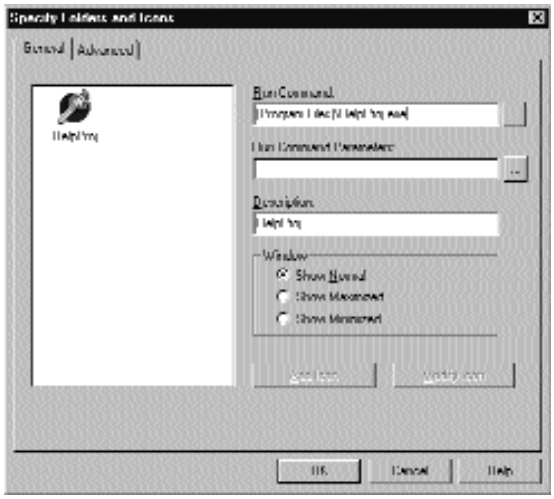
SLIKA 19.23 *Okvir za dijalog Select User Interface Components*

Naredni odeljak iz Setup Checklista je nazvan Make Registry Changes. Ovaj odeljak može biti jednostavan za upotrebu, ali njegovi efekti su dalekosežni jer izmene vršite direktno u System Registryju. (Najčešći razlog za dodavanje Registry elemenata je kreiranje asocijacija između različitih ekstenzija fajlova i Vaše aplikacije.) Zbog toga postupajte oprezno sa dodavanjem Registry podataka i vrednosti. Po definiciji, InstallShield kreira za Vas nekoliko Registry elemenata, primarno da bi registrovao celu putanju aplikacije za korektno pokretanje iz menija Start.

Poslednji odeljak Setup Checklista pre generisanja setup programa je nazvan Specify Folders and Icons. Ovaj odeljak ćete koristiti za navođenje parametara komandne linije aplikacije (bilo EXE putanju i naziv fajla, kao i parametre komandne linije, ili putanju i naziv fajla unapred određenog dokumenta), njeno prikazivanje u meniju Start i druga svojstva ikone menija Start

(recimo, da li aplikacija treba da se pokrene u minimiziranom, maksimiziranom ili normalnom prozoru). Okvir za dijalog Specify Folders and Icons možete videti na slici 19.24.

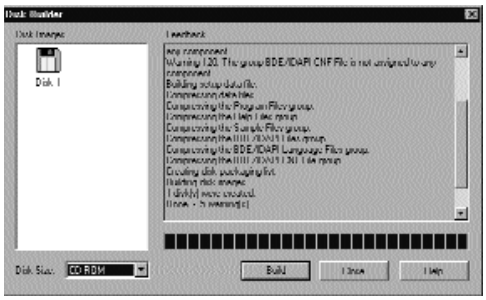
Sada ste spremni da izradite setup program. Naredni odeljak Setup Checklista, nazvan Run Disk Builder, sklapa sva podešavanja i konfiguracije iz prethonih elemenata, spaja ih u kompresovane fajlove sa podacima, a zatim generiše jedan ili više diskova (fajlove koji predstavljaju komponente instalacionih diskova). Ovde zaista počinjete da izrađujete svoj setup program. U okviru za dijalog Disk Builder možete odabrati medij za distribuiranje svog setup programa, a zatim kliknuti kontrolu Build. Od tog trenutka možete se udobno smestiti u stolicu i nadgledati proces, beležeći greške i upozorenja koje InstallShield prikazuje, ukoliko ih ima. Ako se dogode greške, možete da načinite izmene u elementima i zatim ponovo izradite setup program. Na slici 19.25 je prikazan okvir za dijalog Disk Builder posle izrade setup programa koji sadrži nekoliko grešaka



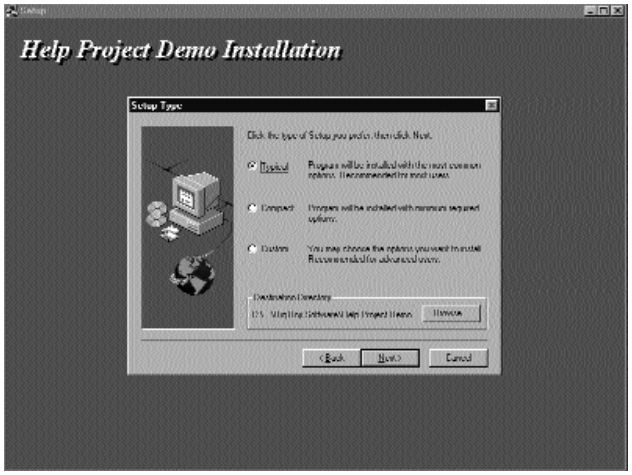
SLIKA 19.24 *Okvir za dijalog Specify Folders and Icons*

Kada ste izradili svoj setup program i eliminisali greške koje je prijavio okvir za dijalog Disk Builder, spremni ste za testiranje setup programa tako što ćete ga izvršiti. Na sreću, InstallShield Vam omogućava da ovo učinite pre nego što zaista izradite diskove. Kada kliknete element Test Run, InstallShield će pokrenuti program i možete da započnete testiranje različitih opcija instaliranja. Ukoliko želite da testirate različite setup tipove u ovom trenutku (pre nego što zaista kreirate instalacione diskove), želećete da deinstalirate softver, upotrebom alata Add/Remove Windows Control Panela. Na slici 19.26 je prikazan novi setup program prilikom izvršavanja.

Kada ste testirali setup program i utvrdili da sve korektno funkcioniše, spremni ste da izradite diskove za instalaciju. Ovo je poslednji element Setup Checklista i, kao što ste očekivali, to je poslednji zadatak koji ćete obaviti u pripremanju setup programa pre konačnog testiranja i dupliranja.



SLIKA 19.25 *Okvir za dijalog Disk Builder*



SLIKA 19.26 *Setup program za primer Help Project*

Administriranje izvornog koda

Kada radite na velikom projektu ili sa timom programera i delite izvorni kod, nemojte raditi sa fajlovima izvornog koda i jednostavno zamenjivati starije verzije novim. Ova standardna procedura može prouzrokovati probleme kada želite da vratite prethodnu verziju izvornog koda, kada neki drugi programer treba da radi sa istim fajlovima, ili je potrebno da zna šta su drugi programeri izmenili.

Delphi 5 obezbeđuje ugrađeno rešenje za ovakve tipove problema kao i za praćenje verzija. Alat je nazvan TeamSource i mora biti instaliran zasebno od Delphi okruženja (zapamtite da je TeamSource na raspolaganju samo u Enterprise verziji Delphija ili ga možete zasebno kupiti). Borland definiše TeamSource kao "alat za upravljanje tokom posla", što je zapravo dobar opis onoga što alat može da obavi za Vas (čak i u jednokorisničkoj situaciji). TeamSource koristi sistem kontrole verzija, koji je nazvan *kontroler*, uključujući jednostavan `Borland.zlib`, koji je deo alata.

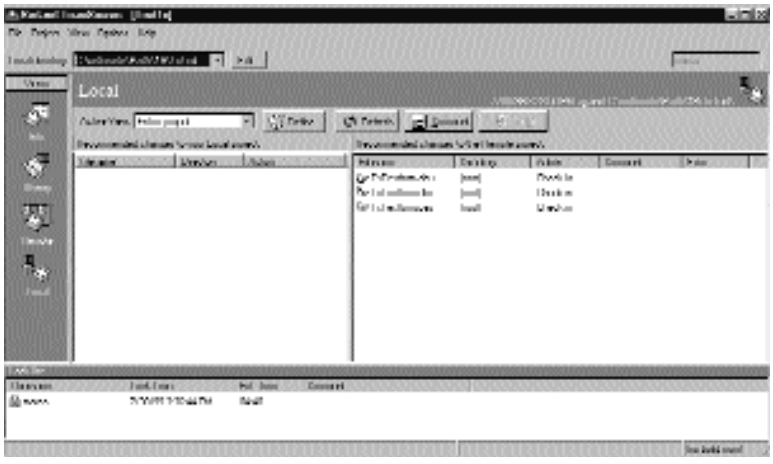
SAVET

TeamSource možete upotrebiti za svoje Delphi programiranje, a takođe i za bilo koji drugi programski jezik ili skup ASCII fajlova. (Fajlovi moraju biti bar zasnovani na tekstu ukoliko želite da poredite razlike i rešite konflikte.) Na primer, TeamSource možete upotrebiti za administriranje HTML fajlovima (velikog) web sajta. ■

TeamSource je prilično komplikovan alat, sa mnogim opcijama i karakteristikama, te ću Vam dati samo rezime njegovih mogućnosti. Osnovna ideja je da će svaki korisnik imati svoju verziju izvornog koda, da može da načini izmene i da može da prosledi izmene u deljeni *udaljeni* skup. Ovo se naziva *paralelna kontrola verzija* (parallel version control), jer mnogi programeri mogu istovremeno da izmene fajlove izvornog koda. Svaki programer radi sa lokalnom kopijom, kao da sam programira projekat. S vremena na vreme korisnik može konsultovati lokalne fajlove sa udaljenom slikom, poslati fajlove koje je korisnik izmenio i dobiti novu kopiju fajlova drugih programera.

Ovaj proces nije jednostavna zamena fajlova. TeamSource vodi računa o razlikama u fajlovima, čuva sve prethodne verzije fajlova izvornog koda, održava dnevnik poruka koje je dodao svaki od korisnika sistema, prosleđuje fajlove i poruke preko elektronske pošte i još mnogo toga. Postoje neke operacije, kao što je prosleđivanje fajlova, koje u jednom trenutku može obaviti samo jedan korisnik. Kada obavite ovakvu operaciju, TeamSource će automatski zaključati ceo projekat (dva korisnika ne mogu istovremeno prosleđivati svoje izmene). Ovo automatsko zaključavanje se uklanja posle kratkog vremena, ukoliko ga ne povećate.

Glavni prozor TeamSourcea koristi liniju menija kao Outlook, gde možete odabrati različite oblasti: udaljeni odeljak, lokalni odeljak (prikazan na slici 19.27), istoriju projekta i opšta podešavanja. Lokalni odeljak je mesto na kome operišete sa fajlovima na lokalnom kompjuteru, čije se ažuriranje održava procesom prosleđivanja. TeamSource upoređuje lokalne informacije sa udaljenim deljenim verzijama, a zatim izmene prikazuje u ovom pogledu, predlažući Vam kako da ih ažurirate (na primer, prosleđivanjem izmene u deljeni odeljak ili preuzimanjem ažurirane verzije koju je načinio neki drugi programer). Iz lokalnog pogleda možete videti razlike između Vaših verzija i verzija koje su poslednje proverene (izborom jedne od verzija koje su na raspolaganju).

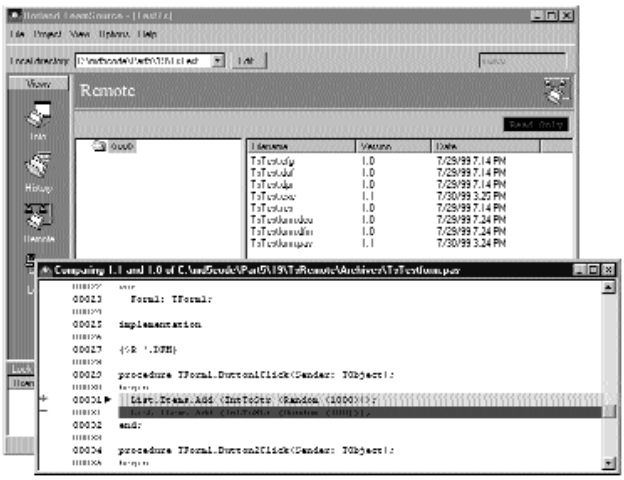


SLIKA 19.27 *Glavni prozor TeamSourcea kada je prikazan pogled Local*

Nasuprot ostalim sistemima za kontrolu verzija, TeamSource Vam omogućava da prijavite Vaše fajlove, ali ne i da ih odjavite. Nije potrebno da obavestite sistem da radite sa fajlom (to jest, da zaključate deljeni fajl da biste sprečili ostale korisnike da ga izmene), jer kada se prijavite, TeamSource će znati koje ste izmene načinili i kako da te izmene uklopi u deljeni resurs na serveru. TeamSource može ovo učiniti upotrebom algoritma "trostrukog upoređivanja" koji upoređuje izvorne fajlove koje ste poslednje kopirali sa servera, izmenjeni izvorni fal na Vašoj lokalnoj mašini i izvorni fajl koji se trenutno nalazi na serveru. Upotrebom ova tri podatka, TeamSource može da razluči ne samo koje izmene ste načinili, već i koje su izmene načinjene nad deljenim fajlom i kako da spoji te izmene bez gubljenja informacija. Jedini slučaj kada ručno morate da spajate izmene u TeamSourceu je kada Vi i neki drugi programer izmenite istu liniju izvornog koda istog izvornog fajla u isto vreme. U ovom slučaju, prva osoba koja obavlja proveru će bez problema spojiti izmene, ali druga osoba koja vrši proveru će biti informisana o "koliziji" i biće upitana da li želi da ručno obavi spajanje. Sa optimističkim zaključavanjem baza podataka, ove kolizije prijavljivanja postaju sve manje i manje verovatne kada se povećava veličina projekta.

Najbolje što pruža TeamSource je to da mnogi programeri mogu da rade na istom projektu, a da ne smetaju jedni drugima. Vi jednostavno radite sa lokalnom kopijom fajlova u Delphiju na uobičajeni način. Ovim je TeamSource manje invazioni od mnogih drugih sistema za kontrolu verzija.

Udaljeni pogled Vam daje stanje projekta. U ovom pogledu možete videti sve izmene u projektu, uraditi revizije i možete uporediti fajlove u listi (to jest, fajlove koji su prijavili različiti programeri tokom vremena). Poređenje fajlova se zasniva na izvornom kodu i prikazuje sve razlike, kao što možete videti na slici 19.28. Možete videti listu History i sve komponente koje su upisane u dnevnik, uključujući i razloge za izmene i za zahteve za zaključavanjem. Još jedan važan koncept je koncept *produkcija* (productions). Produkcija naznačava sistemu da je dati fajl rezultat kompajliranja nekog drugog fajla. Na primer, DCU fajl je produkcija Pascal fajla, a RES fajl je produkcija RC fajla.



SLIKA 19.28 *Prozor za poređenje izvornog koda u TeamSourceu; u pozadini je prikazan pogled Remote*

Kada je potrebno da načinite snimak projekta koji ćete kasnije koristiti, možete kreirati obeležje. Posle toga, moći ćete da načinite lokalnu kopiju projekta bilo kojeg obeležja. Kreiranje obeležja je način da sačuvate stanje svog projekta u određenom trenutku (vreme i datum). Kada pozivate fajlove projekta, možete TeamSourceu naznačiti da pozove fajlove projekta u stanju u kome su bili u određenom trenutku (vreme i datum) za neko obeležje. Usput, pripazite kada pozivate fajlove, jer će se možda izgubiti lokalne izmene koje niste prijavili. Mogu postojati lokalna obeležja, vidljiva samo programeru koji ih je definisao, i globalna, koja određuje administrator.

Još jedno uobičajeno ponašanje za kontrolu verzija je grananje. Razmotrite slučaj procesa programiranja između dva različita programera ili delova tima. Jedan tim može popravljati bagove, dok drugi može dodavati karakteristike koje možda neće odmah biti dostupne korisnicima. Ovo ne bi bilo moguće kada bi svi proveravali osnovu koda. Ipak, TeamSource Vam omogućava da replicirate izvorni kod u dva različita direktorijuma, lokalno i udaljeno. U ovom slučaju, svaki korisnik će ažurirati lokalne izmene u jednom od ova dva direktorijuma, koji sadrže različite grane. Posle nekog vremena, kada je potrebno da dve grupe ponovo obave sinhronizaciju njihovih zasebnih napora, upotrebiće TeamSource za sinhronizovanje dve grane direktojima sa originalnom granom.

U TeamSourceu postoji još mnogo više mogućnosti nego što ovaj kratak opis nagoveštava. Možete načiniti autonomne lokalne kopije da biste eksperimentisali (pozivajući fajl projekta koji Vam je potreban), poništiti prethodno sačuvane verzije i obaviti mnoge druge operacije. Jedini savet koji mogu da Vam dam je da sami isprobate mogućnosti. Dobit koju ćete imati je vredna truda.

Šta je sledeće?

U ovom poglavlju smo proučili detalje o ulozi, definiciji i upotrebi Windows resursa u tradicionalnim aplikacijama i demonstrirali smo kako da ih upotrebimo u Delphi programiranju i za lokalizovanje aplikacija. Zatim smo proučili neke tehnike koje se odnose na štampanje, manipulisanje fajlovima i upotrebu Clipboarda. Zatim, razmotrili smo Delphijevu podršku INI fajlovima i sistemskom Registryju i isprobali smo neke od tehnika. Konačno, razmatrali smo Delphijev online help, alate za instaliranje i TeamSource, dajući kratak opis njihove upotrebe.

Ovim poglavljem završavamo našu diskusiju o praktičnim Delphi tehnikama. Od narednog poglavlja počinjemo da istražujemo još jednu naprednu tehniku koja ima veliku podršku u Delphiju: Internet programiranje i distribuirano programiranje.

POGLAVLJE

20

Internet programiranje

U OVOM POGLAVLJU ĆU DATI UVOD U INTERNET I WEB PROGRAMIRANJE U DELPHIJU, UPOTREBOM NEKIH KOMPONENATA KOJE SU NA RASPOLAGANJU U IDE-U. DOLASKOM ERE INTERNETA, PISANJE PROGRAMA ZA WORLD WIDE WEB JE POSTALO UOBIČAJENO.

ZAPOČEĆEMO PROUČAVANJEM HTML FAJLOVA I IZRADOM NEKOLIKO HTML GENERATORA. NAREDNI KORAK ĆE BITI RAZMATRANJE ACTIVEFORMSA. ZATIM ĆEMO NASTAVITI SA UPOTREBOM DELPHI KOMPONENATA PRIKLJUČAKA (SOCKET), OSTALIH INTERNET KOMPONENATA I TEHNIKA KOJE MOŽETE DA UPOTREBITE ZA AUTOMATSKU OBRADU ELEKTRONSKE POŠTE.

Na kraju, obratićemo pažnju na server, naročito na programiranje server proširenja zasnovanih na uobičajenom ulaznom interfejsu (Common Gateway Interface — CGI), ISAPI i Active Servers stranama (ASP). Naravno, ja ću pokušati da obratim više pažnje na izdavanje baza podataka na Webu upotrebom specifičnih komponenata i alata koji su na raspolaganju u Delphiju. Primetićete da je većina ovih komponenata deo WebBroker tehnologije, koja je na raspolaganju samo u Delphi Enterprise izdanju ili kao zasebni dodatak.

UPOZORENJE

Da biste testirali neke primere ovog poglavlja, potreban Vam je pristup web serveru. Najbolji test je, verovatno, upotreba servera pod Windowsom NT ili Windowsom 2000, ali primere možete isprobati upotrebom Microsoftovog Personal Web Servera, koji je deo Windowsa 98. ■

HyperText Markup Language (HTML)

Jezik HyperText Markup Language, koji je poznatiji pod skraćenicom HTML, veoma je rasprostranjen format hiperteksta na Webu. HTML je format koji Web pretraživači obično čitaju. Takođe, HTML je standard koji je definisao W3C, World Wide Web Consortium, a to je jedno od tela koje kontroliše Internet. Trenutni standard predstavlja HTML 4, mada ga svi pretraživači ne podržavaju u potpunosti. Kada izrađujete web sajt, uvek morate da odaberete najmanji zajednički pristup da biste podržali većinu pretraživača koji se koriste, ukoliko ne ciljate na određenu grupu korisnika od kojih tražite da prihvate određeni pretraživač (što je čest slučaj kada je u pitanju intranet — mreža kompanije ili mala mreža). Ukoliko ne poznajete dobro tagove koji su deo HTML fajlova, možda bi bilo dobro da pročitate dodatak "Format HTML fajlova" da biste ih ukratko upoznali.

Na klijent strani Weba, glavna aktivnost je pretraživanje — čitanje HTML fajlova. U Poglavlju 16 smo već videli kako možete da napišete jednostavan pretraživač umećući Microsoft Internet Controls u Vašu aplikaciju (to jest, upotrebom komponente WebBrowser koja je na raspolaganju na Internet strani Delphijeve palete Components).

Takođe, možete direktno da aktivirate pretraživač koji je instaliran na kompjuteru korisnika, otvarajući HTML stranu pozivanjem metoda `ShellExecute` (definisan je u jedinici ShellApi):

```
ShellExecute (Handle, 'open',
FileName, '', '', sw_ShowNormal);
```

Upotrebom metoda `ShellExecute` možemo jednostavno izvršiti dokument, npr. fajl. Windows će pokrenuti program koji je pridružen HTML ekstenziji, koristeći akciju koja je prosleđena kao parametar (u ovom slučaju open). Možete upotrebiti sličan poziv da biste pogledali web sajt, jednostavnom upotrebom stringa kao što je *'http://www.borland.com'* umesto naziva fajla. U ovom slučaju sistem prepoznaje *http* odeljak zahteva kao zahtevanje web pretraživača, te ga i pokreće.

Na server strani generišete i činite dostupnim HTML strane. Ponekad može biti dovoljno da imate način da napravite statičke strane, izdvajajući s vremena na vreme nove podatke iz tabele baze podataka da biste po potrebi ažurirali HTML fajlove. U ostalim slučajevima je potrebno da dinamički generišete strane na osnovu zahteva korisnika. Ja ću u ovom poglavlju napisati nekoliko primera koji obrađuju prvi slučaj, ali ću dinamičko generisanje ostaviti za naredno poglavlje.

FORMAT HTML FAJLOVA

Ukoliko imate nekakvo znanje o HTML-u ali sa njim ne radite tako često da biste znali sve osnovne elemente, nemojte odustajati jer sledi kratak rezime.

HTML fajlovi su u osnovi ASCII tekst fajlovi. Pored običnog teksta, HTML fajl sadrži mnogo tagova, koji mogu odrediti stil fonta, tip paragrafa ili link na neki drugi HTML fajl ili sliku, između ostalog.

Mnogi tagovi su upareni kao tagovi za početak i tagovi za kraj (tag za kraj je obično isti kao i tag za početak, ali se ispred njega nalazi karakter /) da bi se naznačilo gde počinje stil a gde se završava. Na primer, pišete `<b>important</>` da bi reč important bila napisana masnim slovima, a pišete `<title>Document title</title>` da biste odredili naslov dokumenta.

Neki tagovi nemaju verziju za kraj (ili terminator). Tag `<p>` koji se koristi za odvajanje paragrafa je jedan od ovih tagova. Tag `<p>` je naročito važan jer se razmaci između linija i karakteri za novi red u HTML-u u potpunosti ignorišu. Jedino upotrebom taga `<p>` ili taga `<li>`, ili započinjanjem novog naslova, možete tekst premestiti u novu liniju.

HTML dokument počinje tagom `<html>` i podeljen je na dva dela, koja su označena kao `<head>` i `<body>`. Svakom od ova tri taga je potreban odgovarajući terminator. U head odeljku HTML fajla obično navodite naslov (koji se često prikazuje na naslovnoj liniji pretraživača) i nekoliko drugih generičkih elemenata).

U telu pišete sadržaj fajla, obično počinjući od njegovog vidljivog naslova. Možete imati nekoliko zaglavlja različitih nivoa, koji su označeni tagom `<hX>`, gde X menjate nekim brojem od 1 do 6. Ove tagove slede obični paragrafi (`<p>`), unapred formatirani paragrafi (`<pre>`, stil koji se obično koristi za listinge), različiti tipovi lista i mnogi drugi elementi. Tekst će često sadržati linkove na druge strane ili na delove trenutne strane, upotrebom taga `<a>`.

Još jedan zanimljiv element HTML-a su tabele. Tagovi `<table>` i `</table>` označavaju početak i kraj tabele, a opcioni atribut border prikazuje bordure zadate debljine. Tagovi `<tr>` i `</tr>` uvode i završavaju svaki red, a tagovi `<th>`, `</th>`, `<td>` i `</td>` označavaju ćeliju zaglavlja tabele i ćeliju podataka tabele, respektivno. Broj kolona zavisi od elemenata u svakom redu. Različiti redovi, zapravo, mogu imati različiti broj elemenata.

HTML je tema mnogih knjiga (kako u izdanju Sybexa tako i u izdanjima drugih izdavača), a možete pronaći veliki broj uputstava za HTML samo ukoliko pretražite Web.

Delphijeve HTML Producer komponente

Ukoliko Vaša verzija Delphija sadrži HTML Producer komponente (koje su na raspolaganju na Internet strani Component Palette), možete ih koristiti za generisanje HTML fajlova, a naročito za pretvaranje tabele baze podataka u HTML tabelu. Mnogi programeri veruju da upotreba ovih komponenata ima smisla samo ukoliko pišete proširenje web servera. Mada su predstavljene za ovakvu upotrebu i deo su WebBroker tehnologije, još možete upotrebiti tri od četiri komponente u bilo kojoj aplikaciji u kojoj treba da generišete statički HTML fajl.

Pre nego što se pozabavim primerom HtmlProd, koji pokazuje upotrebu ovih HTML Producer komponenata, dozvolite mi da rezimiram njihovu ulogu:

- Najjednostavnija od HTML Producer komponenata je komponenta PageProducer, koja manipuliše HTML fajlom u koji ste umetnuli specijalne tagove. Prednost ovakvog pristupa je u tome da možete generisati takav fajl upotrebom HTML editora koji Vam se dopada. U vreme izvršavanja, komponenta PageProducer konvertuje specijalne tagove u HTML kod, obezbeđujući Vam direktan metod za izmenu odeljaka HTML dokumenta. Ovi specijalni tagovi imaju osnovni format `<#tagname>`, ali takođe možete navesti imenovane parametre unutar taga. Tagove ćete obrađivati u događaju `OnTag` komponente PageProducer.
- Komponenta DataSetPageProducer je proširenje komponente PageProducer koje automatski zamenjuje tagove koji odgovaraju nazivima polja povezanog izvora podataka.
- Komponenta DataSetTableProducer je obično korisna za prikazivanje sadržaja tabele, upita ili drugih skupova podataka. Ideja je da se proizvede HTML tabela iz skupa podataka, na jednostavan ali ipak fleksibilan način. Komponenta sadrži veoma lep preliminarni prikaz, te možete direktno u vreme dizajniranja videti kako će izgledati HTML izlaz u pretraživaču.
- Komponenta QueryTableProducer je slična prethodnoj komponenti (ovo je zapravo potklasa), ali je specijalno prilagođena za izradu parametarskih upita zasnovanih na unosu sa HTML formulara za pretraživanje. Zbog toga ću razmatrati ovu komponentu tek kada budemo razmatrali programiranje server strane.

Izrada HTML strana

Veoma jednostavan primer upotrebe tagova je kreiranje HTML fajla koji prikazuje polja sa trenutnim datumom ili datumom koji je izračunat kao relativan u odnosu na trenutni, recimo, datum kada nešto ističe. Ukoliko proučite primer HtmlProd pronaći ćete narednu komponentu u glavnom formularu:

```
object PageProducer1: TPageProducer
  HTMLDoc.Strings = (...)
  OnHTMLTag = PageProducer1HTMLTag
end
```

Izvor HTML-a se može naznačiti upotrebom spoljašnjeg fajla (prednost je što taj fajl možete izmeniti a da ne morate ponovo kompajlirati aplikaciju koja ga koristi) ili liste stringova koja se čuva u `HTMLDoc` svojstvu. Ovo je običan HTML fajl koji može sadržati nekoliko specijalnih tagova koji se uvode simbolom #:

```
<HTML><HEAD>
<TITLE>Producer Demo</TITLE>
</HEAD><BODY>
<H1>Producer Demo</H1>
This is a demo of the page produced by the <b><#appname></b>
Application on <b><#date></b>.<p>
<hr>
```

```
The prices in this catalog are valid until <b>
<#expiration days=21></b>.<p>
</BODY></HTML>
```

UPOZORENJE

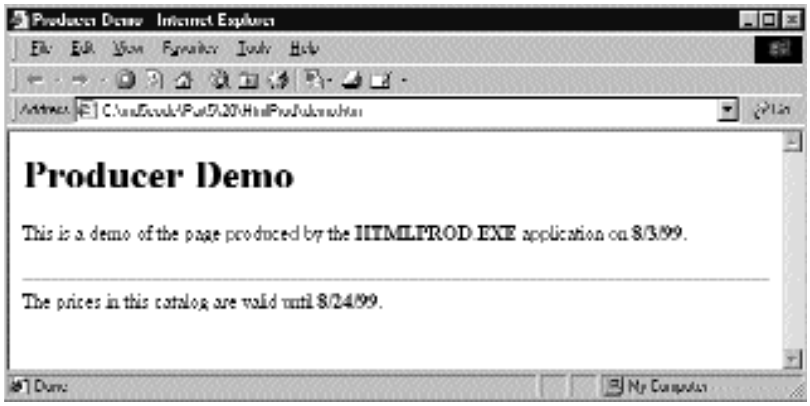
Ukoliko ovaj fajl pripremite upotrebom HTML editora (a to Vam preporučujem), editor može automatski staviti navodnike oko tag parametara, npr. `days="21"`, jer je to neophodno za HTML 4. Komponenta PageProducer u Delphiju 5 sadrži novo svojstvo `StripParamQuotes`, koje se može aktivirati radi uklanjanja ovih dodatnih navodnika kada komponenta čita kod (pre pozivanja obrade događaja `OnHTMLTag`). ■

Kontrola Demo Page jednostavno kopira izlaz komponente PageProducer u svojstvo `Text` Memo komponente sledećim iskazom:

```
Memo1.Text := PageProducer1.Content;
```

Kada pozovete funkciju `Content` komponente PageProducer, funkcija čita ulazni HTML kod, prevodi ga i inicira obradu događaja `OnTag` za svaki specijalni tag. U ovom metodu proveravamo vrednost taga (prosleđenom u parametru `TagString`) i kao rezultat dajemo različit HTML tekst (u parametru `ReplaceText`) formirajući izlaz koji je prikazan na slici 20.1.

```
procedure TFormProd.PageProducer1HTMLTag(Sender: TObject;
  Tag: TTag; const TagString: String; TagParams: TStrings;
  var ReplaceText: String);
var
  nDays: Integer;
begin
  if TagString = 'date' then
    ReplaceText := DateToStr (Now)
  else if TagString = 'appname' then
    ReplaceText := ExtractFilename (Forms.Application.Exename)
  else if TagString = 'expiration' then
  begin
    nDays := StrToIntDef (TagParams.Values['days'], 0);
    if nDays <> 0 then
      ReplaceText := DateToStr (Now + nDays)
    else
      ReplaceText := '<I>{expiration tag error}</I>';
  end;
end;
```



SLIKA 20.1 *Izlaz primera HtmlProd, jednostavna komponenta PageProducer, kada korisnik klikne kontrolu Demo Page*

Naročito obratite pažnju na kod koji smo napisali za konvertovanje poslednjeg taga, *expiration*, koji zahteva parametar. Komponenta PageProducer smešta ceo tekst parametra taga (u ovom slučaju, *days=21*) u string koji je deo liste `TagParams`. Da biste izdvojili iz stringa deo u kome se nalazi vrednost (deo posle znaka jednakosti), možete iskoristiti svojstvo `Values` liste stringova `TagParams` i istovremeno tražiti odgovarajući element. Ukoliko ne može da pronađe parametar, ili njegova vrednost nije ceo broj, DLL prikazuje poruku o grešci.

SAVET

Komponenta PageProducer podržava tagove koje definiše korisnik, a koji mogu biti bilo koji string koji napišete, ali bi prvo trebalo da proučite tagove definisane `TTag` enumeracijom. Moguće vrednosti uključuju `tgLink` (za tag LINK), `tgImage` (za tag IMAGE), `tgTable` (za tag TABLE) i nekoliko drugih vrednosti. Ukoliko kreirate sopstveni tag, kao u primeru PageProd, vrednost parametra `Tag` za obradu HTMLTag će biti `tgCustom`. ■

Izrada strana podataka

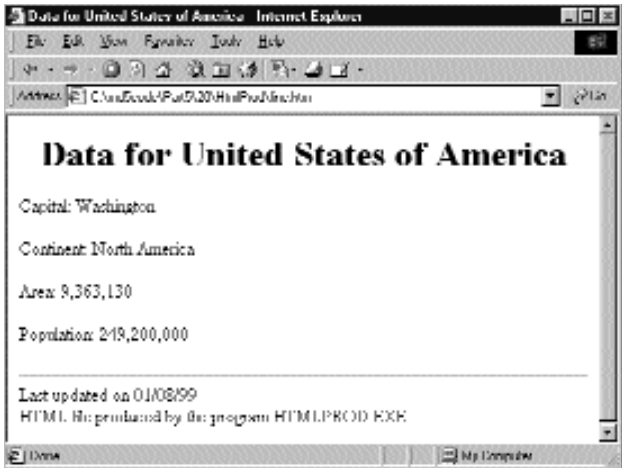
Primer HtmlProd takođe sadrži komponentu DataSetPageProducer, sa sledećim podešavanjima i HTML izvornim kodom:

```
object DataSetPageProducer1: TDataSetPageProducer
  HTMLDoc.Strings = (
    '<HTML><HEAD>'
    '<TITLE>Data for <#name></TITLE>'
    '</HEAD><BODY>'
    '<H1><CENTER>Data for <#name></CENTER></H1>'
    'Capital: <#capital><p>'
    'Continent: <#continent><p>'
    'Area: <#area><p>'
    'Population: <#population><p>'
    '<HR>'
    'Last updated on <#date><br>'
    'HTML file produced by the program <#program>'
```

```
    '</BODY></HTML>')
  OnHTMLTag = DataSetPageProducer1HTMLTag
  DataSet = Table1
end
```

Jednostavnom upotrebom tagova koji odgovaraju nazivima polja povezanog skupa podataka (uobičajena tabela baze podataka Country.DB), program automatski uzima vrednost polja trenutnog sloga i automatski ga zamenjuje, dajući izlaz koji je prikazan na slici 20.2. U izvornom kodu programa koji se odnosi na ovu komponentu, zapravo, ne postoji referenca na podatke baze podataka:

```
procedure TFormProd.BtnLineClick(Sender: TObject);
begin
  Memo1.Clear;
  Memo1.Text := DataSetPageProducer1.Content;
  BtnSave.Enabled := True;
end;
procedure TFormProd.DataSetPageProducer1HTMLTag(
  Sender: TObject; Tag: TTag; const TagString: String;
  TagParams: TStrings; var ReplaceText: String);
begin
  if TagString = 'program' then
    ReplaceText := ExtractFilename (Forms.Application.Exename)
  else if TagString = 'date' then
    ReplaceText := DateToStr (Date);
end;
```



SLIKA 20.2 *Izlaz primera HtmlProd za kontrolu Print Line*

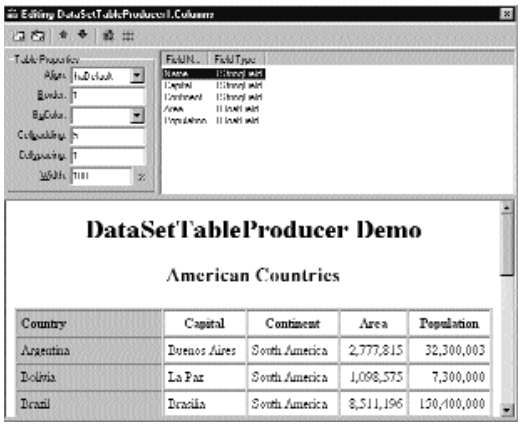
Izrada HTML tabela

Poslednja kontrola primera HtmlProd je Print Table. Ova kontrola je povezana sa komponentom DataSetTableProducer, koja poziva funkciju `Content` komponente i kopira njen rezultat u svojstvo `Text` Memo komponente. Jednostavnim povezivanjem svojstva `DataSet` komponente DataSetTableProducer na `Table1` možete generisati standardnu HTML tabelu. Zapravo, unapred je određeno da komponenta generiše 20 redova, što je naznačeno svojstvom `MaxRows`. Ukoliko želite da prikažete sve redove tabele, možete odrediti vrednost –1, a to nije dokumentovana vrednost.

SAVET

Komponenta DataSetTableProducer počinje od trenutnog sloga pre nego od prvog sloga tabele. To znači da sledeći put kada kliknete kontrolu Print Table, neće se prikazati nijedan slog na izlazu. Dodavanje poziva `First` metodu tabele pre nego što se pozove metod Content komponente, rešiće problem. ■

Da biste izlaz ove komponente učinili potpunijim, možete obaviti dve različite operacije. Prva je obezbeđivanje `Header` i `Footer` informacije, za generisanje HTML zaglavlja i završnih elemenata, i dodavanje zaglavlja (svojstvo `Caption`) HTML tabeli. Druga je prilagođavanje same tabele upotrebom vrednosti koja se navodi u svojstvima `RowAttributes`, `TableAttributes` i `Columns`. Editor svojstava kolona, koji je inače osnovni editor komponente, omogućava Vam da odredite većinu svojstava, istovremeno obezbeđujući veoma lep preliminarni prikaz, kao što možete videti na slici 20.3. Pre upotrebe ovog editora možete podesiti svojstva za polja tabele, upotrebom Fields Editora. Evo kako, na primer, možete formatirati izlaz polja o populaciji i površini upotrebom separatora hiljada.



SLIKA 20.3 *Editor svojstva Columns komponente DataSetTableProducer Vam obezbeđuje preliminarni prikaz konačne HTML tabele (ako je aktivna tabela baze podataka)*

Postoje tri tehnike koje možete upotrebiti za prilagođavanje HTML tabele i vredi predstaviti svaku od njih:

- Možete upotrebiti svojstvo `Column` komponente Table Producer za određivanje svojstava, kao što su tekst i boja naslova, ili boja i poravnanje u ćelijama ostatka kolone. Vrednosti za primer možete videti u prethodnom listingu.

- Možete upotrebiti `TField` svojstva, naročito ona koja se odnose na izlaz. U primeru sam podesio svojstvo `DisplayFormat` objekta polja `Table1Content` na `###,###,###`. Ovo je pristup koji treba da koristite ukoliko želite da odredite izlaz svakog polja. Možete otići još dalje i umetnuti HTML tagove u izlaz polja.
- Možete obraditi događaj `OnFormatCell` komponente DataSetTableProducer da biste prilagodili izlaz za štampač. U ovom događaju možete podesiti različite atribute kolone jedinstveno za svaku ćeliju, ali možete prilagoditi i izlazni string (koji se čuva u parametru `CellData`) i umetnuti HTML tagove. Ovo je nešto što ne možete učiniti upotrebom svojstva `Columns`.

U primeru sam koristio obradu ovog događaja da bih tekst kolona Population i Area prikazao masnim slovima na crvenoj pozadini kada su vrednosti velike (izuzev ukoliko nije u pitanju zaglavlje reda). Sledi kod:

```
procedure TFormProd.DataSetTableProducer1FormatCell(
  Sender: TObject; CellRow, CellColumn: Integer;
  var BgColor: THTMLBgColor; var Align: THTMLAlign;
  var VAlign: THTMLVAlign; var CustomAttrs, CellData: String);
begin
  if (CellRow > 0) and
    (((CellColumn = 3) and (Length (CellData) > 8)) or
    ((CellColumn = 4) and (Length (CellData) > 9))) then
  begin
    BgColor := 'red';
    CellData := '<b>' + CellData + '</b>';
  end;
end;
```

Ostatak koda je sumiran podešavanjima komponente Table Producer:

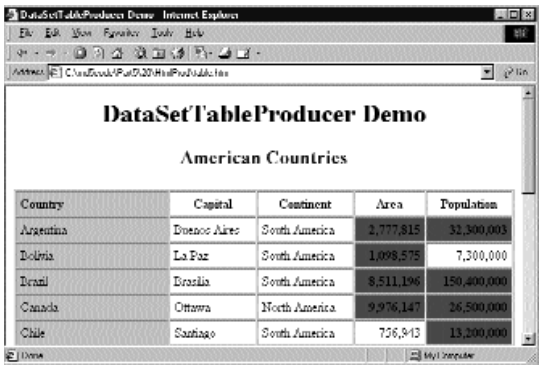
```
object DataSetTableProducer1: TDataSetTableProducer
  Caption = '<h2>American Countries</h2>'
  Columns = <
    item
      BgColor = 'Silver'
      FieldName = 'Name'
      Title.Align = haLeft
      Title.BgColor = 'Silver'
      Title.Caption = 'Country'
    end
    item
      FieldName = 'Capital'
    end
    item
      FieldName = 'Continent'
    end
    item
      Align = haRight
      FieldName = 'Area'
    end
    item
      Align = haRight
      FieldName = 'Population'
```

```
    end>
  Footer.Strings = (
    '<hr><i>Produced by EmplProd</i>'
    '</body></html>')
  Header.Strings = (
    '<html><head>'
    '<title>DataSetTableProducer Demo</title>'
    '</head><body>'
    '<h1><center>DataSetTableProducer Demo</center></h1>')
  MaxRows = -1
  DataSet = Table1
  TableAttributes.Border = 1
  TableAttributes.CellPadding = 5
  OnFormatCell = DataSetTableProducer1FormatCell
end
```

Izlaz ovog programa možete videti na slici 20.4. Preporučujem Vam da proučite izvorni kod HTML fajla koji generiše ovaj program, tako da možete shvatiti bogatstvo njegovog izlaza i stoga prednosti upotrebe ove komponente.

Upotreba stilova

Poslednje inkarnacije HTML-a sadrže veoma moćan mehanizam za odvajanje sadržaja od prezentacije: kaskadni stilovi (cascading style sheets — CSS). Upotrebom stilova možete odvojiti formatiranje HTML-a (boja, fontova, veličina fontova i tako dalje) od pravog teksta koji se prikazuje (sadržaja strane). Ovaj pristup čini Vaš kod mnogo fleksibilnijim, a Vaš web pretraživač će se lakše ažurirati. Uz to, možete odvojiti zadatak da sajt učinite grafički privlačnim (posao web dizajnera) od automatskog generisanja sadržaja (posao programera). Stilovi su složena tehnika, u kojoj obezbeđujete vrednosti za glavne tipove HTML odeljaka i za specijalne "klase" (koje nemaju ništa zajedničko sa OOP-om). Ponoviću, pogledajte HTML reference za više detalja.



SLIKA 20.4 *Izlaz kontrole Print All primera HtmlProd, koji je zasnovan na komponenti DataSetTable Producer*

Kako možemo ažurirati generisanje tabele u primeru HtmlProd da bismo uključitli stilove? Jednostavno, link na stilove koje želimo da koristimo u svojstvu Header druge komponente DataSetTableProducer, možemo obezbediti sledećom linijom:

```
<link rel = "stylesheet" type = "text/css" href = "test.css">
```

Zatim možemo ažurirati kod obrade događaja `OnFormatCell` sledećom akcijom (umesto dve linije kojima se menja boja i dodaje tag za masna slova):

```
CutomAttrs := 'class="higlight"';
```

Stil koji sam obezbedio (TEST.CSS, na raspolaganju je u izvornom kodu primera) definiše stil za *isticanje* (higlight), a to su masna slova na crvenoj pozadini koja su dobijena kodiranjem prve komponente DataSetTableProducer.

Prednost ovog pristupa je u tome da grafički umetnici sada mogu da izmene CSS fajl i da Vašoj tabeli mogu da daju lepši izgled, a da ne moraju da menjaju kod. Kada želite da obezbedite mnogo elemenata za formatiranje, upotreba stila može da smanji ukupnu veličinu HTML fajla. Ovo je važan element koji može da smanji vreme potrebno za preuzimanje HTML fajlova.

Izdavanje statičkih baza podataka na Webu

Kada ste saznali kako da načinite strane, možete jednostavno dodati linkove sa jedne na drugu stranu i generisati niz međusobno povezanih HTML fajlova, koji predstavljaju deo web sajta. Postoje situacije u kojima pisanje programa koji pretražuje bazu podataka i generiše fajlove predstavlja najbolji način za izdavanje podataka baze podataka na web sajtu. Slične tehnike možete upotrebiti i u sledećim situacijama:

KADA SE PODACI NE MENJAJU ČESTO Katalog koji se ažurira mesečno i nedeljno je dobar primer. Čak i kada možete da ažurirate sajt svake večeri, ovo je i dalje tehnika koju možete upotrebiti. (Za informacije u realnom vremenu ovo svakako nije dobar pristup!)

KADA JE KOLIČINA PODATAKA OGRANIČENA I MANJA OD RASPOLOŽIVOG PROSTORA NA WEB SAJTU Ovo je očigledno, ali formatirani HTML izlaz može da zauzme mnogo više prostora od originalnih fajlova baze podataka. Ukoliko za generisanje HTML-a iz baze podataka koristite program na server strani (kao što su programi koje razmatram u narednom poglavlju), možda će Vam biti potrebno manje prostora na disku za web sajt. Imajte na umu da pripremanje svih HTML fajlova unapred obično daje bolje performanse (bolje vreme odgovora servera na web zahteve i manju upotrebu memorije za procesiranje zahteva) nego generisanje podataka tokom rada.

KADA JE BROJ NAČINA ZA PRETRAŽIVANJE OGRANIČEN Ukoliko postoje tri ili četiri očigledna načina za pretraživanje (glavni i dva ili tri unakrsna načina), možete ih sve statički generisati. U suprotnom, unakrsni HTML fajlovi će biti mnogo veći od fajlova koji sadrže podatke, a vreme potrebno da se generišu može biti ogromno.

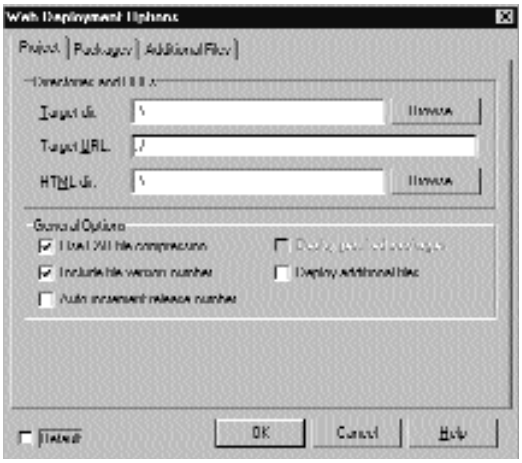
Čak i kada se ovi uslovi delimično odnose na Vaše potrebe, možete da razmislite o upotrebi više različitih pristupa. Možete imati deo podataka i fajlove za kretanje koji se periodično generišu, CGI i ISAPI aplikaciju na sajtu, a možete i da korisnicima omogućite slobodno pretraživanje i praćenje manje korišćenih puteva. Kasnije u ovom poglavlju ćemo videti kako da to postignete.

Za sada ću obratiti pažnju na sasvim drugačiju tehnologiju: izdavanje ActiveX kontrola i ActiveFormsa na web sajtu.

ActiveForms na web stranama

U Poglavlju 16 ste naučili kako da koristite Delphijevu tehnologiju ActiveForms za kreiranje novih ActiveX kontrola. U tom poglavlju sam istakao da je ActiveForm zapravo ActiveX kontrola koja se zasniva na formularu. Borlandova dokumentacija često implicira da ActiveForms treba da se koristi na HTML stranama, ali na web strani možete da koristite bilo koju ActiveX kontrolu.

U osnovi, svaki put kada kreirate ActiveX biblioteku, Delphi aktivira elemente menija Project➥Web Deployment Options i Project➥Web Deploy. Prvi element menija Vam omogućava da naznačite kako i gde se šalju odgovarajući fajlovi. Kao što se vidi na slici 20.5, u tom okviru za dijalog možete odrediti direktorijum servera za smeštanje ActiveX komponente, URL ovog direktorijuma i direktorijum servera za smeštanje HTML fajlova (koji će imati reference na ActiveX biblioteku upotrebom URL-a koji navedete).



SLIKA 20.5 *Okvir za dijalog Web Deployment Options*

Takođe, možete odrediti upotrebu kompresovanih CAB fajlova, koji mogu sadržati OCX fajl i druge pomoćne fajlove, kao što su paketi, čineći lakšim i bržim prosleđivanje aplikacije korisniku. Kompresovani fajl, zapravo, znači brže preuzimanje. Upotrebom opcija koje su prikazane na slici 20.5, Delphi generiše HTML fajl i CAB fajl za projekat XClock (koji je izrađen u Poglavlju 16) u istom direktorijumu. Otvaranjem ovog HTML fajla u Internet Exploreru (ne zaboravite, Netscape ne podržava ActiveX kontrole) dobija se izlaz prikazan na slici 20.6.

SLIKA 20.6 *Kontrola XClock u primeru HTML strane*

UPOZORENJE

Ponekad kada učitate HTML stranu koja se referiše na ActiveX kontrolu, sve što ćete dobiti je crvena X oznaka koja Vas obaveštava da preuzimanje kontrole nije bilo uspešno. Postoje različita objašnjenja ovog problema. Prvo, Internet Explorer mora biti pravilno podešen tako da omogućava preuzimanje kontrola i (ukoliko kontrola nije potpisana) da smanji nivo sigurnosti. Drugo, mogu iskrsnuti i drugi problemi kada kontrola zahteva DLL ili paket koji nije deo preuzetog CAB fajla. Treće, crvena oznaka se može pojaviti kada se ne podudara broj verzije – ili se može desiti da se prikaže starija verzija kontrole. To je zato što čak i kada ponovo izradite kontrolu, Internet Explorer može odlučiti da koristi keširanu verziju (koja se čuva u direktorijumima `windows/occache` ili `windows/downloaded program files`). Možete upotrebiti informaciju o verziji i druge odgovarajuće tehnike da biste izbegli treći problem. Nije naročito profesionalno, ali poslednja mogućnost koju možete upotrebiti je uklanjanje kopije fajla iz keša kada sve drugo ne daje rezultate. Zbog bagova koji postoje u Internet Exploreru 3 i 4, ukoliko je starija verzija ActiveX kontrole još uvek učitana u memorijski keš pretraživača, pretraživač neće preuzeti niti instalirati noviju verziju kontrole, čak i kada tag reference objekta HTML-a jasno pokazuje da je u pitanju novija verzija. Takođe, fajl keširane kontrole može biti zaključan, te da bi fajl mogao pravilno da se prosledi potrebno je da zatvorite Internet Explorer i da zatim prosledite fajl. ■

Pored toga što ću Vam pokazati kako da prosledite kontrolu XClock sa web strane, ja sam kreirao primer XForm1 da bih Vam pokazao probleme koji mogu nastati sa obradama događaja ActiveFormsa koji su pomenuti na kraju Poglavlja 16, u odeljku "Unutrašnjost ActiveFormsa". Kako se događaji formulara izvoze kao događaji kontrole, ne smete direktno da obrađujete ove događaje, već je potrebno da dodate kod unapred određenim obradama koje obezbeđuju ActiveForms. Na primer, ukoliko dodate obradu `OnPaint` događaja formulara i napišete sledeći kod, taj kod se nikada neće izvršiti:

```
procedure TFormX1.FormPaint(Sender: TObject);
begin
  Canvas.Brush.Color := clYellow;
  Canvas.Ellipse(0, 0, Width, Height);
end;
```

Ukoliko želite da nešto iscrtate na pozadini formulara, potrebno je da izmenite odgovarajuću obradu koju instalira ActiveForm Wizard:

```
procedure TFormX1.PaintEvent(Sender: TObject);
begin
```

```
  Canvas.Brush.Color := clBlue;
  Canvas.Rectangle (20, 20,
    ClientWidth - 20, ClientHeight - 20);
  if FEvents <> nil then FEvents.OnPaint;
end;
```

Alternativa je da smestite okvir, panel ili neku drugu komponentu na površinu formulara i obradite događaje *te* komponente. U primeru XForm1 ja sam jednostavno dodao komponentu PaintBox, a ispod nje se nalazi istaknuta komponenta da bi komponenta PaintBox bila vidljiva.

Uloga ActiveX formulara na web strani

Pre nego što se pozabavimo narednim primerom, važno je razmotriti ulogu ActiveX formulara koji je smešten unutar web strane. U osnovi, smeštanje formulara na web stranu odgovara omogućavanju korisniku da preuzme i izvrši Windows aplikaciju. Postoji još nešto što se odigrava. Vi preuzimate izvršni fajl i pokrećete ga. (To je razlog što ActiveX tehnologija pokreće mnoga pitanja vezana za sigurnost.)

Jednostavan primer može istaći ovakvu situaciju. Za naredni primer ja sam generisao novi ActiveForm, na koji sam dodao kontrolu i oznaku, a zatim sam napisao sledeći kod za `OnClick` događaj kontrole:

```
procedure TXFormUser.Button1Click(Sender: TObject);
var
  UserName: string;
  Size: Cardinal;
begin
  Size := 128;
  SetLength (UserName, Size);
  GetUserName (PChar(UserName), Size);
  Label1.Caption := UserName;
end;
```

Ovaj metod jednostavno poziva Windows API funkciju `GetUserName` i njegov efekat sigurno nije zapanjujući, jer će se ime korisnika prikazati u oznaci. Ipak, ovaj primer ističe nekoliko važnih stvari (koje se odnose kako na ActiveForms tako i na ActiveX kontrole):

- U ActiveX kontroli ili formularu možete pozvati bilo koju Windows API funkciju (što znači da korisnik koji prikazuje web stranu mora imati Windows na svom kompjuteru) ili određene Windows API kompatibilne biblioteke.
- ActiveX može pristupiti sistemskim informacijama kompjutera, recimo korisničkom imenu, strukturi direktorijuma i tako dalje. To je razlog zbog kojeg, pre preuzimanja ActiveX kontrole, web pretraživači proveravaju da li ActiveX kontrola ima korektan sertifikat ili potpis. (Trebalo bi zapamtite da ova signatura samo identifikuje autora kontrole i da modul nema grešku od kada je autor izdao kontrolu; ovo ni na koji način ne govori da li je bezbedno koristiti kontrolu.)

Dakle, mogao bih da nastavim da navodim razloge, ali smatram da sam Vam ukazao na problem. ActiveX kontrole i ActiveForms su sjajni alati, naročito za lokalnu mrežu. Na Internetu neki korisnici ne žele da imaju ActiveX kontrole.

ActiveForm sa više strana

Za poslednji primer ću pretvoriti postojeći program u ActiveForm. Videli smo da postoje tri standardna pristupa u programiranju složenog programa: MDI, više prioritetnih ili neprioritetnih formulara i formulari sa više strana. Poslednji pristup najviše odgovara programiranju složenog ActiveForma.

Ukoliko želite da postojeći formular pretvorite u ActiveX formular, postoji nekoliko pristupa koje možete slediti. Najjednostavniji način je, verovatno, selektovanje svih komponenata u originalnom programu, izrada šablona komponente na osnovu komponenata (tako da kopirate svojstva komponente i njihove obrade događaja) i, na kraju, prebacivanje komponenata na novi ActiveForm. Drugi pristup je da postojeći formular smestite unutar ActiveForma. Ovo nije naročito komplikovan zadatak, a ima veliku prednost jer ne morate ništa da izmenite u originalnom izvornom kodu, čak ni metode koji obrađuju događaje formulara. (Pošto izvorni kod nije deo ActiveForma, njegovi događaji nisu povezani sa spoljašnjim svetom.)

Da bih Vam praktično pokazao ovaj pristup, ja sam uzeo primer WizardUI iz Poglavlja 8, dodao sam mu web mogućnosti, a zatim sam ga smestio unutar ActiveForma. Novi primer je nazvan XWebWiz. Ukoliko se prisetite originalnog primera, znate da je prikazivao informacije o web sajtovima. Ja sam svojstva oznaka i lista koje su se odnosile na web sajtove podesio na sledeći način:

```
object Label2: TLabel
  Cursor = crHandPoint
  Caption = 'Main site: www.inprise.com'
  Font.Color = clBlue
  Font.Style = [fsUnderline]
  OnClick = LabelLinkClick
end
```

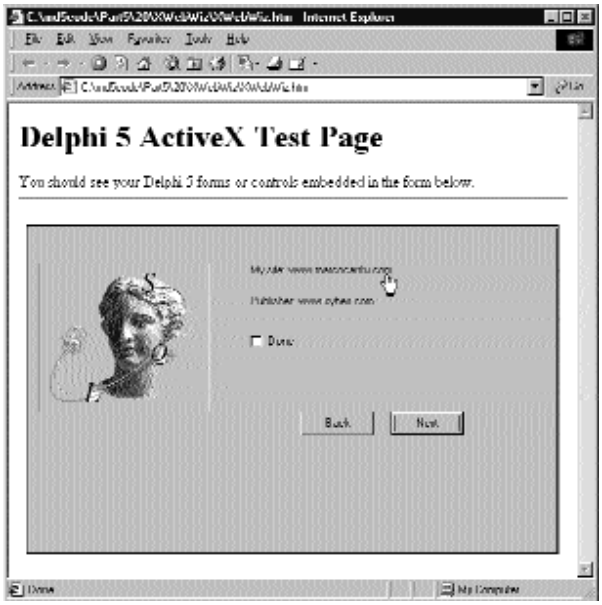
Plavi podvučeni tekst će izgledati kao tipični link unutar pretraživača, a kursor u obliku ruke upotpunjava sliku. Da bismo aktivirali ove linkove, možemo da izdvojimo URL web sajta iz zaglavlja oznake ili elementa liste i da pozovemo API funkciju `ShellExecute` da bismo došli na URL. Da bih program pretvorio u ActiveForm, ja sam jednostavno kreirao novi ActiveForm, dodao jedinicu originalnog formulara projektu i napisao sledeću obradu događaja `OnCreate` za aktivni formular:

```
uses
  WizForm;
procedure TXWizForm.FormCreate(Sender: TObject);
begin
  WizardForm := TWizardForm.Create (Self);
  WizardForm.Parent := Self;
  WizardForm.Align := alClient;
  WizardForm.BorderStyle := bsNone;
  WizardForm.Show;
end;
```

Podešavanjem svojstva `Parent` aktivnog formulara, njegovim poravnavanjem i uklanjanjem bordure, postojeći program će prekriti celu površinu aktivnog formulara, te nećemo moći da razlikujemo ova dva formulara. Primer izlaza možete videti na slici 20.7.

SAVET

Naravno, ActiveForm sa više strana može da se bazira na okvirima, kao što smo to videli u Poglavlju 8. ■



SLIKA 20.7 *Izlaz kontrole XWebWiz na HTML strani. Selektovanjem linkova možete preći na odgovarajući web sajt*

Određivanje svojstava za XArrow

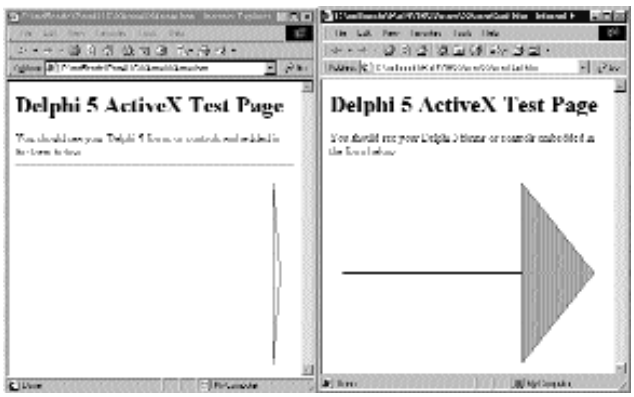
ActiveForm sadrži nekoliko svojstava koja možete podesiti kada ga koristite unutar razvojnog okruženja, a obična ActiveX kontrola ima još više takvih svojstava. Na primer, ukoliko želite da podesite svojstva u HTML fajlu koji sadrži kontrolu, možete upotrebiti specijalni Param tag, ali kontrola mora da podržava specijalni interfejs koji je poznat kao `IPersistPropertyBag`.

Počevši od Delphija 4, `IPersistPropertyBag` podrška je ugrađena, obezbeđujući podršku za sva svojstva ActiveX kontrole ili ActiveForma. Za primer sam uzeo opcije Web Deploy za XArrow kontrolu izrađenu u Poglavlju 16. Zatim sam izmenio automatski generisan HTML fajl pomoću tri Param taga:

```
<OBJECT
  classid="clsid:5551EB27-0AC6-11D2-B9F1-004845400FAA"
  codebase="./XArrow.cab"#version=1,0,0,0
  width=350
  height=250
  align=center
  hspace=0
  vspace=0
>
<Param Name="ArrowHeight" Value="100">
```

```
<Param Name="Filled" Value="-1">
<Param Name="FillColor" Value="111829">
</OBJECT>
```

Na slici 20.8 možete da uporedite određeni izlaz i prilagođeni izlaz kontrole



SLIKA 20.8 *Upotrebom Param taga možemo odrediti vrednosti za svojstva ActiveX kontrole u HTML fajlu koji sadrži kontrolu. Dve kopije programa prikazuju unapred određeni izlaz i prilagođeni izlaz*

Programiranje priključaka upotrebom Delphija

Do sada smo u ovom poglavlju videli kako da prezentujemo statičke HTML fajlove na web sajtu i kako da HTML strane učinimo bogatijim umetanjem Windows programa na strane (u formi ActiveX kontrola).

Sada ću više pažnje posvetiti Internet programiranju, naročito povezivanju koje obezbeđuju Delphijeve komponente za priključke (socket components), koje se zasnivaju na TCP/IP-u i Windows Socketsu niskog nivoa. Pre nego što se pozabavimo osnovama priključaka, ja ću nabrojati alternativne pristupe koje možete upotrebiti za Internet programiranje, a koje ću detaljnije obraditi u narednih nekoliko odeljaka:

- Delphi Socket Components obezbeđuje dobar interfejs za direktnu upotrebu Windows Sockets API-ja, implementirajući neke Vaše protokole. Upotreba Delphi komponenata priključaka višeg nivoa je mnogo lakša od upotrebe API-ja niskog nivoa.
- Za standardne protokole takođe možete koristiti FastNet Tools VCL komponente (NetMasters), koje su deo Delphija, ili možete potražiti slične kontrole drugih nezavisnih programera.
- WinInet (Windows Internet) biblioteka je kolekcija servisa višeg nivoa koju obezbeđuje Microsoft. Ovi servisi čine programiranje HTTP i FTP programa veoma jednostavnim.

Osnove programiranja priključaka

Da biste razumeli opis Socket komponenata u Delphijevom Help fajlu, a takođe i da biste mogli da pratite opise primera koji se nalaze u ovoj knjizi, potrebno je dobro razumeti brojne termine koji se odnose na Internet, a naročito na priključke.

Srž Interneta je protokol za kontrolu transfera/Internet protokol (Transmission Control Protocol/Internet Protocol, ili skraćeno TCP/IP), kombinacija dva različita protokola koji zajedno funkcionišu da bi obezbedili vezu preko Interneta (a takođe mogu obezbediti vezu u lokalnoj mreži). Ukratko, IP je odgovoran za definisanje i rutiranje datagrama (*datagrams* — Internet jedinica prenosa) i za određivanje šeme adresiranja. TCP je odgovoran za servise prenosa višeg nivoa. Pored TCP-a postoji i drugi, manje poznati protokol: UDP (User Datagram Protocol — korisnički datagram protokol).

Konfigurisanje lokalne mreže: IP adrese

Ukoliko na raspolaganju imate lokalnu mrežu, moći ćete da testirate sledeće programe; u suprotnom, jednostavno možete upotrebiti isti kompjuter i kao server i kao klijent. U tom slučaju, kao što sam ja učinio u svojim primerima, možete upotrebiti adrese `127.0.0.1` (ili localhost), što je sasvim sigurno adresa aktuelnog kompjutera. Ukoliko je Vaša mreža složena, zatražite od svog administratora mreže da Vam odredi odgovarajuće IP adrese. Ukoliko želite da podesite jednostavnu mrežu na nekoliko slobodnih kompjutera, možete jednostavno sami podesiti IP adrese, 32-bitni broj koji je obično predstavljen putem četiri svoje komponente (nazvane oktetima — octet) koje su odvojene tačkom. Iza ovih brojeva se nalazi komplikovana logika, a prvi oktet označava klasu adrese.

Postoje određene IP adrese koje su rezervisane za interne mreže koje nisu registrovane. Internet ruteri će ignorisati opsege ovih adresa, te ih možete slobodno koristiti a da ne utičete na mrežu. Opseg "slobodnih" IP adresa je od `192.168.0.0` do `192.168.255.0` i može se koristiti za eksperimentisanje na mreži koja ima manje od 255 mašina.

Nazivi lokalnih domena

Kako se IP adresa mapira na naziv? Na Internetu, klijent program pretražuje vrednosti na domain serveru. Međutim, moguće je postojanje lokalnog (*hosts*) fajla, tekstualnog fajla koji lako možete izmeniti da biste obezbedili lokalno mapiranje. Možete pogledati Hosts.SAM fajl (koji se nalazi u `Windows` direktorijumu) da biste videli primer i da biste eventualno promenili naziv fajla u HOSTS, ali ne menjajte ekstenziju, da biste aktivirali lokalno mapiranje.

Da li treba da koristite IP ili host naziv u svojim programima? Host nazive je lakše zapamtiti i ne moraju se menjati ukoliko se promeni IP adresa (iz bilo kog razloga). S druge strane, IP adrese ne zahtevaju nikakvo razrešavanje, dok se host nazivi moraju razrešiti (operacija koja zahteva dosta vremena ukoliko se odvija na Webu).

TCP portovi

Svaka TCP veza se odvija preko *porta*. Port je predstavljen 16-bitnim brojem. IP adresa i TCP port zajedno određuju neku Internet vezu, ili *priključak* (socket) — precizniji termin. Različiti procesi koji se izvršavaju na istoj mašini ne mogu koristiti isti priključak — isti port.

Neki TCP portovi imaju standardnu upotrebu za određene protokole visokog nivoa i za servise. Drugim rečima, trebalo bi da koristite brojeve tih portova kada implementirate te servise, a da ih u drugim slučajevima izbegavate. Evo kratke liste:

Protokol	Port
HTTP (Hypertext Transfer Protocol)	80
FTP (File Transfer Protocol)	21
SMTP (Simple Mail Transfer Protocol)	25
POP3 (Post Office Protocol, verzija 3)	110
Telnet	23

Services fajl (još jedan tekstualni fajl koji je sličan fajlu Hosts) prikazuje standardne portove koje koriste servisi. Možete i Vi dodati svoj element listi, dajući svom servisu naziv koji odaberete. Klijent priključci uvek određuju broj porta ili naziv servisa server priključka na koji žele da se priključe.

Protokoli visokog nivoa

Ja sam do sada mnogo puta koristio termin *protokol*, ali šta to tačno znači? Protokol je skup pravila o kojima moraju da se dogovore klijent i server da bi mogli da odrede tok komunikacije. Internet protokole niskog nivoa, kakav je TCP/IP, obično implementira operativni sistem. Međutim, termin *protokol* se takođe koristi za Internet Standard Protocols visokog nivoa (kakvi su HTTP, FTP ili SMTP). Ovi protokoli su definisani u standardnim dokumentima koji se mogu pronaći na Webu na adresi `http://www.internic.net.site`.

Ukoliko želite da implementirate korisničku komunikaciju, možete definisati sopstveni (verovatno jednostavni) protokol, skup pravila koja određuju koje zahteve klijent može poslati serveru i kako može odgovoriti na različite moguće zahteve. Primer ovakvog protokola ćemo videti kasnije. Protokoli transfera su na višem nivou od protokola transmisije, jer su oni apstrakt transportnog mehanizma koji obezbeđuje TCP/IP. Ovim su protokoli nezavisni ne samo od operativnog sistema i hardvera, već i od mreže.

Veze priključaka

Kako započinjete komunikaciju preko priključka? Prvo počinje da se izvršava server program, ali on jednostavno čeka zahtev klijenta. Klijent program zahteva vezu naznačavajući server na koji želi da se priključi. Kada klijent pošalje zahtev, server može da prihvati povezivanje, pokrećući specifičan server priključak, koji se povezuje na klijent priključak.

Da bi se podržao ovaj model, postoje tri različita tipa priključaka za povezivanje:

- *Klijent povezivanje* (client connections) je inicirano od strane klijenta i povezuje lokalni klijent priključak sa udaljenim server priključkom. Klijent priključci moraju da opišu server na koji žele da se priključe, navodeći host naziv ili IP adresu i njen port.
- *Osluškivači povezivanja* (listening connections) su pasivni server priključci koji očekuju klijenta. Kada klijent načini novi zahtev, server generiše novi priključak koji se dodeljuje toj vezi i zatim se vraća u osluškivanje. Server osluškivači

priključaka moraju da naznače port koji predstavlja servis koji obezbeđuje. (Zapravo, klijent će se povezati preko tog porta.)

- *Server povezivanje* (server connections) je povezivanje koje aktiviraju serveri kada prihvate zahtev klijenta.

Ovi različiti tipovi povezivanja su važni samo za uspostavljanje veze od klijenta ka serveru. Kada se veza uspostavi, obe strane su slobodne da čine zahteve i šalju podatke drugoj strani.

Delphi komponente priključaka

Delphi sadrži komponente priključaka klijenta i servera. Namena komponenata priključaka je da učine jednostavnim čitanje i pisanje informacije preko TCP/IP veze.

U Component Paletti postoje samo dve komponente priključaka, `TServerSocket` i `TClientSocket`. Obe klase su izvedene iz osnovne klase `TAbstractSocket`, apstraktne klase za sve komponente priključaka, koja je definisana u jedinici `ScktComp`. Svojstva klase `TAbstractSocket` opisuju IP adresu priključka i servis koji obezbeđuje ili traži. Ne koriste sve izvedene klase sva svojstva klase `TAbstractSocket`. Na primer, server priključci ne čine dostupnim IP adresu, jer je implicitno čita sistem koji izvršava aplikaciju.

Upotrebom komponente priključka možete odrediti host (domaćina) i servis upotrebljavajući jedno od sledećeg:

- Svojstvo Host označava naziv domena i servis određenog sistema (koriste ga klijent priključci).
- Svojstvo `Address`, string sa četiri broja u standardnoj Internet notaciji (koriste ga klijent priključci).
- Svojstvo `Port`, broj koji određuje port.
- Svojstvo `Service`, string koji označava naziv servisa.

Ipak, ovo nisu samo klase koje obezbeđuju podršku za priključke. Za povezivanje sa Windows priključcima postoji i klasa `TCustomWinSocket` koja ima tri potklase: `TServerWinsocket`, `TClientWinSocket` i `TServerClientWinSocket`. Ove klase obavijaju obradu oko Windows veze priključaka, a koriste ih glavne komponente priključaka za upravljanje Windows Socket API poziva i za čuvanje informacija o komunikacionoj vezi priključka. Klasa `TCustomWinsocket` se odnosi na hendl priključka, koji je naznačen svojstvom `SocketHandle`, a odnosi se i na hendl sakrivenog prozora koji se koristi za prihvatanje poruka priključka, koji je naznačen svojstvom `Handle`.

Klase izvedene iz TCustomWinSocket predstavljaju različite tipove veze: `TClientWinSocket` predstavlja vezu klijenta, `TServerWinSocket` predstavlja vezeosluškivanja,`TServerClientWinSocket` predstavalja server vezu. Na kraju, postoji i specifična potklasa `TStream`, klasa `TWinSocketStream` i specifična potklasa `TThread`, klasa `TServerClientThread`.

Upotreba priključaka

Posle ove teorije, pogledajmo nekoliko jednostavnih primera. Prvi se nalazi u direktorijumu `Sock1` i sačinjavaju ga aplikacije Server1 i Client1. Server sadrži formular sa sledećim komponentama:

```
object ServerSocket1: TServerSocket
  Active = True
  Port = 50
  ServerType = stNonBlocking
  OnClientConnect = ServerSocket1ClientConnect
  OnClientDisconnect = ServerSocket1ClientDisconnect
  OnClientRead = ServerSocket1ClientRead
end
```

Sav kod aplikacije se odnosi na događaje ove komponente, jer program ne obezbeđuje nikakvu specijalnu interakciju sa korisnikom. Ipak, server sadrži tri liste za prikazivanje statusa, za poruke koje šalje klijent i za dnevnik događaja. Na primer, kada se klijent poveže, server dodaje adresu klijenta u dnevnik:

```
procedure TForm1.ServerSocket1ClientConnect(Sender: TObject;
  Socket: TCustomWinSocket);
begin
  lbLog.Items.Add ('Connected: ' +
    Socket.RemoteHost + ' (' +
    Socket.RemoteAddress + ')' );
  PostMessage (Handle, wm_RefreshClients, 0, 0);
end;
```

Primetićete da događaj `OnClientConnect` označava prvu situaciju da server zna da je klijent povezan. Upotrebom svojstva `Socket`, koje se odnosi na `TCustomWinSocket` niskog nivoa, server može da prati ko pokušava da se poveže. Na kraju ovog i drugih događaja ja želim da ažuriram listu veza, upotrebom `ActiveConnections` svojstva servera. Ipak, u obradi događaja `OnClientConnect` ova lista se još uvek ne ažurira, te ja šaljem poruku formularu da bih odložio operaciju:

```
const
  wm_RefreshClients = wm_User;
procedure TForm1.RefreshClients; // message wm_RefreshClients
var
  I: Integer;
begin
  lbClients.Clear;
  for I := 0 to ServerSocket1.Socket.ActiveConnections - 1 do
    with ServerSocket1.Socket.Connections [I] do
      lbClients.Items.Add (
        RemoteAddress + ' (' + RemoteHost + ')');
end;
```

Sličan kod se izvršava kada se klijent odjavi sa servera:

```
procedure TForm1.ServerSocket1ClientDisconnect(Sender: TObject;
  Socket: TCustomWinSocket);
begin
  lbLog.Items.Add ('Disconnected: ' +
    Socket.RemoteHost + ' (' +
```

```
    Socket.RemoteAddress + ')' );
  PostMessage (Handle, wm_RefreshClients, 0, 0);
end;
```

Na kraju, kada klijent pošalje neke informacije serveru (zapisuje preko priključka), server može pročitati poruku pozivanjem funkcije `ReceiveText`. Ovu operaciju čitanja bi trebalo da obavite samo kada su podaci dostupni — to jest, kada se inicira događaj `OnClientRead`. Imajte na umu da je ovo *destruktivno* čitanje: informacije koje se izdvajaju iz toka se u potpunosti uklanjaju iz toka. Evo koda:

```
procedure TForm1.ServerSocket1ClientRead(Sender: TObject;
  Socket: TCustomWinSocket);
begin
  // read from the client
  lbMsg.Items.Add (Socket.RemoteHost + ': ' +
    Socket.ReceiveText);
end;
```

Sada možemo preći na klijent stranu aplikacije, koja sadrži formular sa klijent komponentom priključka sa sledećim svojstvima:

```
object ClientSocket1: TClientSocket
  Active = False
  Address = '127.0.0.1'
  ClientType = ctNonBlocking
  Port = 50
  OnConnect = ClientSocket1Connect
  OnDisconnect = ClientSocket1Disconnect
end
```

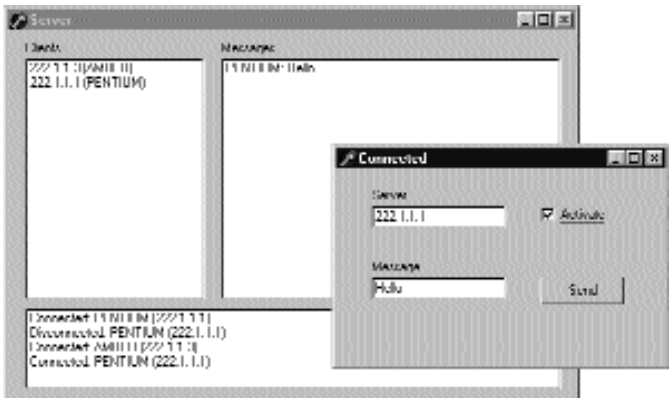
Klijent formular ima više interakcije. Sadrži dva polja za izmene i polje za potvrdu. U prvom polju za izmene možete uneti adresu servera na koji želite da se povežete (da biste promenili unapred određenu vrednost koja je prethodno prikazana), koristeći polje za potvrdu da biste aktivirali ili deaktvirali vezu priključka:

```
procedure TForm1.cbActivateClick(Sender: TObject);
begin
  if not ClientSocket1.Active then
    ClientSocket1.Address := EditServer.Text;
  ClientSocket1.Active := cbActivate.Checked;
end;
```

Kada se povezujete ili odjavljujete, program jednostavno ažurira zaglavlje formulara. U drugo polje za izmene možete uneti poruku koja se šalje serveru i možete kliknuti kontrolu da biste poslali poruku:

```
procedure TForm1.btnSendClick(Sender: TObject);
begin
  ClientSocket1.Socket.SendText (EditMsg.Text);
end;
```

Primetićete da ovaj primer programa ne proverava da li je veza aktivna pre nego što je upotrebi, što može dovesti do grešaka. Na slici 20.9 možete videti primer klijenta i primer servera. Kako server naznačava, postoji druga kopija klijent aplikacije koja se izvršava na drugom kompjuteru i koja je povezana.



SLIKA 20.9 *Klijent i server aplikacije primera Sock1, koje prikazuju upotrebu komponenata priključaka*

Upotreba priključaka uz korisničke protokole

Izuzev ukoliko ne želite da šaljete i primate samo jednostavne tekstualne poruke, možete definisati neka pravila komunikacije između klijenta i servera. Skup pravila komunikacije se obično naziva protokol. U osnovi, server može primati različite zahteve i, u zavisnosti od tipa zahteva i od toga da li se zahtev može obraditi ili ne, odgovriti klijentu.

Server program Sock2 primera prihvata četiri tipa zahteva: listanje direktorijuma, fajl bitmape, tekstualni fajl i izvršavanje programa na serveru. Kada server pošalje fajl nazad, njegov odgovor treba da naznači šta će poslati i da naznači stvarne podatke. Jedini metod koji je izmenjen u odnosu na primer Sock1 je procedura `ServerSocket1ClientRead`, koja započinje izdvajanjem pet početnih karaktera iz teksta koji dobija od klijenta koji šalje komandu:

```
strCommand := Socket.ReceiveText;
lbLog.Items.Add ('Client: ' + Socket.RemoteAddress +
  ': ' + strCommand);
// extract the file name (all commands have 5 characters)
strFile := Copy (strCommand, 6, Length (strCommand) - 5);
```

Kod zavisi od inicijalne komande definisane protokolom (u ovom slučaju `EXEC!` za izvršavanje fajla na serveru, `TEXT!` za dobijanje tekstualnog fajla, `BITM!` za dobijanje fajla bitmape ili `LIST!` za dobijanje listinga direktorijuma). Sledi kod za dve od četiri alternative:

```
// send back a text file
if Pos ('TEXT!', strCommand) = 1 then
begin
  if FileExists (strFile) then
  begin
    strFeedback := 'TEXT!';
    Socket.SendText (strFeedback);
    Socket.SendStream (TFileStream.Create (
      strFile, fmOpenRead or fmShareDenyWrite));
  end
  else
  begin
```

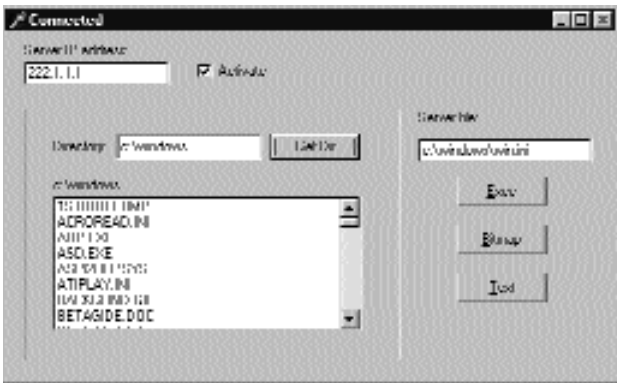
```
    strFeedback := 'ERROR' + strFile + ' not found';
    Socket.SendText (strFeedback);
  end;
end
// send back a directory listing
else if Pos ('LIST!', strCommand) = 1 then
begin
  if DirectoryExists (strFile) then
  begin
    strFeedback := 'LIST!';
    Socket.SendText (strFeedback);
    FileListBox1.Directory := strFile;
    Socket.SendText (FileListBox1.Items.Text);
  end
  else
  begin
    strFeedback := 'ERROR' + strFile + ' not found';
    Socket.SendText (strFeedback);
  end;
end
else
begin
  strFeedback := 'ERROR' + 'Undefined command: ' + strCommand;
  Socket.SendText (strFeedback);
end;
```

Za listinge direktorijuma sam koristio nevidljivu komponentu FileListBox. Za slanje tekstualnog fajla koristio sam metod `SendStream`, kreirajući novi tok. Prednost je što se ne mora ukloniti privremeni tok, jer `SendStream` postaje vlasnik toka i uklanja ga kada završi rad.

Program šalje nazad više delova informacija jedne za drugim. Ovo će stvoriti nekoliko problema na strani klijenta, jer se sve informacije dobijaju preko jednog toka. Ipak, server odgovara hederom od pet karaktera koji možemo upotrebiti za određivanje sadržaja ostatka toka. Posle prihvatanja ovih hedera, klijent aplikacija podešava status polje tako da zna koji tip informacija pristiže. Drugim rečima, klijent program koji implementiramo je veoma jednostavna mašina, a to je tipična tehnika za programiranje priključaka. Klijent aplikacija ima pet mogućih stanja, koja su navedena u pobrojanom tipu:

```
type
TCliStatus = (csIdle, csList, csBitmap, csText, csError);
```

Ovo je tip podataka koji se koristi za `CliStatus` polje formulara. Formular sadrži dva polja za izmene koja se odnose na direktorijum ili fajl koji od servera može zahtevati korisnik. Kada korisnik upotrebi kontrolu Get Dir, klijent program prosleđuje serveru naziv direktorijuma koji je naznačen prvim poljem za izmene. Server će vratiti spisak fajlova koji klijent program čuva u listi. U ovom trenutku korisnik može selektovati jedan od fajlova iz liste, klijent program će ga kopirati, kao i celu putanju, u drugo polje za izmene. Tekst drugog polja za izmene koriste preostale tri kontrole, Exec, Bitmap i Text, koje šalju dodatne zahteve serveru. Na slici 20.10 možete videti primer glavnog formulara klijent programa pošto je dobijen sadržaj direktorijuma.



SLIKA 20.10 *Formular programa Client2 pošto je server poslao spisak fajlova direktorijuma*

Srž programa se nalazi u metodu `ClientSocket1Read`, koji inicira priključak kada postoje podaci koje treba pročitati. Metod se prvo koristi za dobijanje zaglavlja, koje označava tip podataka koji stižu do programa, i za određivanje pravilnog statusa programa:

```
case CliStatus of
  // look for data to receive
  csIdle:
  begin
    Socket.ReceiveBuf (Buffer, 5);
    strIn := Copy (Buffer, 1, 5);
    if strIn = 'TEXT!' then
      CliStatus := csText
    else if strIn = 'BITM!' then
      CliStatus := csBitmap
    // .. and so on
```

Pošto ne dobijamo sve podatke, događaj se ponovo inicira odmah posle toga, a ovoga puta smo spremni da prihvatimo podatke. Evo još dve grane **case** iskaza:

```
  // get a directory listing
  csList:
  begin
    ListFiles.Items.Text := Socket.ReceiveText;
    cliStatus := csIdle;
  end;
  // read a bitmap file
  csBitmap:
    with TFormBmp.Create (Application) do
    begin
      Stream := TMemoryStream.Create;
      Screen.Cursor := crHourglass;
      try
        while True do
        begin
          nReceived := Socket.ReceiveBuf (Buffer, sizeof (Buffer));
          if nReceived <= 0 then
            Break
```

```
        else
          Stream.Write (Buffer, nReceived);
        // delay (200 milliseconds)
        Sleep (200);
      end;
      // reset and load the temporary file
      Stream.Position := 0;
      Image1.Picture.Bitmap.LoadFromStream (Stream);
    finally
      Stream.Free;
      Screen.Cursor := crDefault;
    end;
    Show;
    cliStatus := csIdle;
  end;
```

Za učitavanje bitmape ja jednostavno premeštam podatke u `Buffer` (koji je deklarisan kao niz `[0..9999] of Char`), a zatim iz bafera u memorijski tok, koji se kasnije učitava u komponentu Image sekundarnog formulara. Pošto se tok podataka može usporiti, program sadrži odlaganje od 200 milisekundi posle svakog čitanja podataka. Za razliku od operacija čitanja fajla, petlja se ne završava kada ima manje podataka za čitanje nego što je zahtevano, već samo kada više nema podataka za čitanje. (U slučaju greške, vrednost koja se dobija metodom `ReceiveBuff` je `–1`.)

Blokirajuće, neblokirajuće višeprocesne veze

Čitanje i pisanje putem priključaka se može odvijati asinhrono, tako da ne blokira izvršavanje ostalog koda Vaše mrežne aplikacije. Ovo se naziva *neblokirajuće povezivanje* (nonblocking connection), a to je ono što ćemo sada uraditi, ostavljajući unapred određenu vrednost `ctNonBlocking` za svojstvo `ClientType` i za svojstvo `ServerType` dve komponente priključaka. Neblokirajuće veze čitaju i pišu asinhrono: događaji klijenta `OnRead` i `OnWrite` i događaji servera `OnClientRead` ili `OnClientWrite` informišu Vaš priključak kada drugi kraj veze pokušava da čita ili piše podatke.

Kao alternativu ovom asinhronom pristupu možete upotrebiti blokirajuće veze, kada Vaša aplikacija čeka da se obavi čitanje ili pisanje pre nego što nastavi izvršavanje naredne linije koda. U ovom slučaju morate napisati kod u nizu na obe strane, jer u suprotnom događaji neće biti inicirani. Kada koristite blokirajuće veze, na serveru morate upotrebiti proces, a u opštem slučaju ćete i na klijentu koristiti proces. Na server strani alternativna vrednost za svojstvo `ServerType` je `stThreadBlocking`.

Kao što sam ranije pomenuo, kada pišete kod preocesa koji radi sa blokirajućom vezom, možete upotrebiti klasu `TWinSocketStream` za obavljanje čitanja i pisanja. Možete upotrebiti metod `WaitForData` klase `TWinSocketStream` da sačekate dok priključak sa druge strane veze ne bude spreman za pisanje. Takođe, možete kreirati klasu toka priključka i odrediti vreme, tako da ukoliko se veza izgubi, nećete čekati u nedogled.

Slanje podataka baze podataka preko priključka veze

Upotrebom tehnika koje smo do sada upoznali možemo napisati aplikaciju koja premešta slogove baze podataka preko priključka. Ideja je napisati deo za unos podataka i deo za čuvanje podataka. Klijent aplikacija će sadržati jednostavan formular za unos podataka i koristiće tabelu baze podataka sa string poljima Company, Address, State, Country, Email i Contact, polje u kome se čuvaju podaci u pokretnom zarezu za ID kompanije (nazvano CompID).

NAPOMENA

Slanje slogova baze podataka preko priključka je upravo ono što MIDAS i komponente priključka čine. Ovo se razmatra u narednom poglavlju. ■

Klijent program koji sam izradio radi sa tabelom čija se struktura čuva u trenutnom direktorijumu. (Odgovarajući kod možete videti u obradi događaja `OnCreate`.) Osnovni metod na strani klijenta je obrada događaja `OnClick` kontrole Send All, koji šalje sve nove slogove serveru. Novi slogovi se određuju proverom toga da li slog sadrži validnu vrednost polja CompID. Ovo polje, zapravo, ne određuje korisnik svojim unosom, već vrednost polja određuje server aplikacija kada su podaci poslati.

Za sve nove slogove klijent program pakuje informacije polja u listu stringova, koristeći strukturu *FieldName=FieldValue*, koja se dobija upotrebom svojstva *Values* liste stringova. String koji odgovara celoj listi se šalje serveru. U ovom trenutku program se zaustavlja u očigledno beskonačnoj petlji:

```
 // save database data in a string list
Data := TStringList.Create;
table1.First;
while not Table1.Eof do
begin
  // if the record is still not logged
  if Table1CompID.IsNull or (Table1CompId.AsInteger = 0) then
  begin
    lbLog.Items.Add ('Sending ' + Table1Company.AsString);
    Data.Clear;
    // create strings with structure "FieldName=Value"
    for I := 0 to Table1.FieldCount - 1 do
      Data.Values [Table1.Fields[I].FieldName] :=
        Table1.Fields [I].AsString;
    // send the record
    ClientSocket1.Socket.SendText (Data.Text);
    // wait for response
    fWaiting := True;
    while fWaiting do
      Application.ProcessMessages;
  end;
  Table1.Next;
end;
```

Program zauvek čeka… ili čeka dok obrada neke druge poruke ne promeni vrednost `False` polja `fWaiting`. Ovo se dešava kada server pošalje informacije koje govore da je slog primljen ili kada korisnik klikne kontrolu Stop. Metod `btnSendAllAlick` automatski povezuje na server na početku i odjavljuje na kraju.

Sada obratimo pažnju na server. Ovaj program sadrži tabelu baze podataka, koja se čuva u lokalnom direktorijumu, sa dva nova polja koja su dodata tabeli klijent aplikacije: LoggedBy, string polje; i LoggedOut, polje podataka. Vrednosti ova dva dodatna polja server automatski određuje kada dobije podatke, kao što određuje i vrednost polja CompID. Sve ove operacije se obavljaju u metodu `ServerSocket1ClientRead` posle raspakivanja podataka koje je primio klijent:

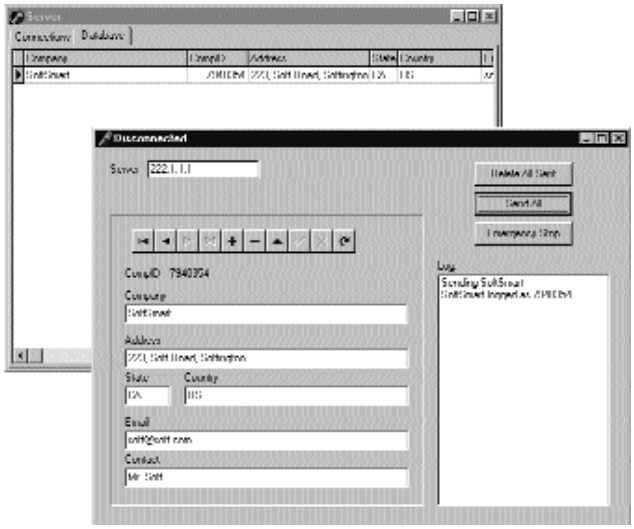
```
// read from the client
strCommand := Socket.ReceiveText;
// reassemble the data
Data := TStringList.Create;
try
  Data.Text := strCommand;
  // new record
  Table1.Insert;
  // set the fields using the strings
  for I := 0 to Table1.FieldCount - 1 do
    Table1.Fields [I].AsString :=
      Data.Values [Table1.Fields[I].FieldName];
  // complete with random ID, sender, and date
  Table1CompID.AsInteger := GetTickCount;
  Table1LoggedBy.AsString := Socket.RemoteAddress;
  Table1LoggetOn.AsDateTime := Date;
  Table1.Post;
  // get the value to return
  strFeedback := Table1CompID.AsString;
  // send results back
  lbLog.Items.Add (strFeedback);
  Socket.SendText (strFeedback);
finally
  Data.Free;
end;
```

Izuzev činjenice da neki podaci mogu da se izgube, ne postoji nikakav problem kada su polja u drugačijem redosledu i ukoliko se ne poklapaju, jer se podaci čuvaju u strukturi *FieldName=FieldValue*. Posle dobijanja svih podataka i njihovog slanja u loklanu tabelu, server šalje nazad klijentu ID kompanije. Klijent program, posle slanja sloga, prelazi u *mod čekanja*, situaciju koja se menja kada server pošalje informacije:

```
procedure TForm1.ClientSocket1Read(Sender: TObject;
  Socket: TCustomWinSocket);
begin
  if fWaiting then
  begin
    Table1.Edit;
    Table1CompId.AsString := Socket.ReceiveText;
    Table1.Post;
    lbLog.Items.Add (Table1Company.AsString +
      ' logged as ' + Table1CompId.AsString);
    fWaiting := False;
  end;
end;
```

Kada primi informacije, klijent program čuva ID kompanije, kojim se slog označava kao poslat. Ukoliko korisnik izmeni slog, ne postoji način da se ažurirane informacije pošalju serveru. Da biste ovo postigli, možete dodati izmenjeno polje tabeli baze podataka klijenta i učiniti da server proveri da li prima novo polje ili izmenjeno polje. Kada prima izmenjeno polje, server ne sme dodati novi slog već mora ažurirati postojeći.

To je jedan od mnogih dodataka koje možete pridružiti programu da biste ste ga učinili upotrebljivim u realnom okruženju. Postojeći kod programa i prethodni primeri priključaka bi trebalo da Vam obezbede sve što Vam je potrebno za obavljanje sličnog zadatka. Ja sam sebe ograničio na ovu verziju aplikacije, kao što je prikazano na slici 20.11. Primetićete da server program sadrži dve strane, jednu sa uobičajenim dnevnikom, a drugu sa komponentom DBGrid koja prikazuje aktuelne podatke tabele baze podataka servera.



SLIKA 20.11 *Klijent i server programi primera priključka baze podataka (DbSock)*

Internet protokoli

Posle razmatranja generisanja HTML fajlova, upotrebe ActiveX tehnologije za web sajtove i komponenata priključaka niskog nivoa, spremni smo da se pozabavimo poslednjom temom ovog poglavlja, upotrebom Internet protokola višeg nivoa. Ovo je, zapravo, najjednostavniji deo ovog poglavlja, jer protokoli visokog nivoa koje ćemo obraditi mogu da se programiraju upotrebom komponenata visokog nivoa ili upotrebom API-ja.

Kao što je već pomenuto, Delphi dobijate sa kolekcijom Internet komponenata NetMasters. Ove komponente obezbeđuju potpuno rešenje, koristeći alternativni pristup u odnosu na Delphijeve komponente priključaka. Mnogo interesantnije komponente serije FastNet Tools su komponente koje implementiraju specifične protokole, protokole NMFinger, NMNNPT, NMFTP, NMHTTP, NMPOP3, NMSMTP. Ove komponente se obično koriste u klijent aplikacijama za povezivanje na

specifične servere. Delphi dobijate sa primerima koje odmah možete koristiti, a koji koriste većinu NetMasters komponenata.

NAPOMENA

Ove komponente nezavisnih programera su već instalirane u Delphi IDE-u, ali one nisu jedino moguće rešenje. Postoji mnogo besplatnih Delphi komponenata koje obezbeđuju implementaciju Internet protokola. Jedno od najinteresantnijih rešenja je Winshoes otvoreni projekat koji vodi Čad Hover (Chad Hower). Više informacija i aktuelne komponente možete preuzeti sa sajta `www.pbe.com/Winshoes`. ■

Slanje i primanje pošte

Verovatno najuobičajenija operacija koju obavljate putem Interneta jeste slanje i primanje elektronske pošte. Ne postoji mnogo razloga za pisanje kompletne aplikacije za obradu elektronske pošte, jer su neki postojeći programi više nego dovoljni. Zbog toga ja nemam nameru da napišem program za poštu koji je opšte namene. Neke primere možete pronaći i direktorijumu Delphi Internet Demos.

Sem kreiranja aplikacije za poštu opšte namene, šta još možemo učiniti sa komponentama i protokolima za poštu? Postoje mnoge mogućnosti koje sam pokušao da razvrstam u tri oblasti:

AUTOMATSKO GENERISANJE PORUKA Aplikacija koju ste napisali može sadržati okvir About za slanje poruke o registrovanju odeljenju za marketing, ili određeni element menija za slanje zahteva za tehničku podršku. Možete čak odlučiti da aktivirate tehničku pomoć svaki put kada se dogodi izuzetak. Drugi sličan zadatak bi bio automatizovanje slanja poruka listi ljudi ili generisanje automatskih poruka sa Vašeg web sajta (primer koji ću Vam pokazati pri kraju poglavlja).

UPOTREBA PROTOKOLA POŠTE ZA KOMUNICIRANJE SA KORISNICIMA KOJI SU SAMO POVREMENO NA VEZI Kada je potrebno da prebacite podatke između korisnika koji nisu stalno na vezi, možete napisati aplikaciju na serveru za sinhronizovanje među njima, a svakom korisniku možete dati klijent aplikaciju za interakciju sa serverom. Alternativa je upotreba postojeće server aplikacije, kao što je server pošte, i pisanje dva specijalizovana programa koja koriste protokole pošte. Podaci koji se šalju na ovaj način će biti formatirani na dva specijalna načina, tako da možete koristiti određene adrese elektronske pošte za ove poruke (ne Vaše primarne adrese elektronske pošte).

Slanje poruka programu za poštu

Najjednostavnija tehnika za automatizovanje generisanja poruka je upotreba postojeće aplikacije za poštu, dodavanjem poruke u odeljak za slanje pošte. Upotrebom API funkcije ShellExecute lako možete poslati poruku `Outlook Expressu` (ili bilo kom programu za poštu koji je registrovan kao osnovni u Windowsu, mada postoji nekoliko izuzetaka).

Da bih testirao ovu tehniku, ja sam pripremio jednostavan formular sa dva polja za izmene i Memo poljem za unos. Kada kliknete kontrolu, kreira se string sa svim informacijama o poruci, a zatim se poruka šalje, jednostavnim izvršavanjem stringa koji ima prefiks mailto. Evo koda kontrole Send primera MailGen:

```
uses
  ShellApi;
procedure TForm1.BtnSendClick(Sender: TObject);
var
  strMsg: string;
  I: Integer;
begin
  // set the basic information
  strMsg := 'mailto:' + EditAddress.Text +
    '?Subject=' + EditSubject.Text +
    '&Body=';
  // add first line
  if Memo1.Lines.Count > 1 then
    strMsg := strMsg + Memo1.Lines [0];
  // add subsequent lines separated by the newline symbol
  for I := 1 to Memo1.Lines.Count - 1 do
    strMsg := strMsg + '%0D%0A' + Memo1.Lines [I];
  // send the message
  ShellExecute (Handle, 'open', pChar (strMsg),
    '', '', SW_SHOW);
end;
```

Da biste prikazali telo poruke u više linija, svaku liniju možete odvojiti oznakom za prelazak u novi red (carriage return i line feed koji se u Delphiju obično označavaju sa #13 i #10). Ove vrednosti bi trebalo da eksplicitno budu dodate i ispred njih bi trebalo da stoji simbol %, kao što to zahteva URL. Ovo kodiranje možete dobiti upotrebom komponente NMURL.

WinInet API

Kada je potrebno da koristite FTP i HTTP protokole, kao alternativu upotrebi specifičnih VCL komponenata, možete upotrebiti jednostavan API visokog nivoa koji obezbeđuje Microsoft u WinInet DLL-u. Ova biblioteka je deo operativnog sistema, a možete je preuzeti sa Microsoftovog web sajta. Ova biblioteka u osnovi implementira FTP i HTTP protokole na vrhu Windows API priključaka.

Jednostavnom upotrebom tri poziva — `InternetOpen`, `InternetOpenURL` i `InternetReadFile` — možete dobiti fajl koji ogovara URL-u, sačuvati ga u lokalnoj kopiji i analizirati. I drugi jednostavni metodi se mogu koristiti za FTP. Savetujem Vam da proučite izvorni kod Delphi jedinice, koja je deo SDK Helpa koji dobijate uz Delphi.

Za primer upotrebe HTTP protokola odlučio sam da napišem veoma specifičnu aplikaciju za pretraživanje. Program se jednostavno povezuje sa Yahoo web sajtom, traži ključnu reč i kao rezultat daje prvih sto sajtova koje pronađe. Umesto da rezultat prikaže u HTML fajlu, program izdvaja samo URL-ove odgovarajućih sajtova. Dakle, program pokazuje dve tehnike odjednom: dobijanje web strane i prevođenje HTML koda.

Posle malo testiranja primetio sam da je WinInet funkcijama potrebno dosta vremena za izvršavanje, jer moraju da sakupe informacije sa Weba. Zbog toga sam odlučio da program implementiram upotrebom procesa u pozadini za obradu. Ovaj pristup takođe ima prednost što može da započne više pretraga odjednom. Klasa procesa koju koristi aplikacija WebFind kao ulaz prihvata URL koji se traži, u promenljivoj `strUrl`:

```
type
  TFindWebThread = class(TThread)
  protected
    strAddr, strStatus: string;
    procedure Execute; override;
    procedure AddToList;
    procedure ShowStatus;
  public
    strUrl: string;
  end;
```

Klasa sadrži dve izlazne proceudre, procedure `AddToList` i `ShowStatus`, koje se pozivaju iz metoda `Synchronize`. (Pogledajte Poglavlje 17 za više detalja o procesima.) Kod ova dva metoda šalje rezultate glavnom formularu, dodajući liniju u Memo komponentu i menjući svojstvo `SimpleText` statusne linije. Ključni metod procesa je metod `Execute`. Pre nego što ga pogledamo, dozvolite mi da Vam pokažem kako se glavnim formularom aktivira proces:

```
procedure TForm1.BtnFindClick(Sender: TObject);
var
  FindThread: TFindWebThread;
begin
  // create suspended, set initial values, and start
  FindThread := TFindWebThread.Create (True);
  FindThread.FreeOnTerminate := True;
  FindThread.strUrl :=
    'http://search.yahoo.com/bin/search?p=' +
    EditSearch.Text + '&n=100&h=s&b=1';
  FindThread.Resume;
end;
```

URL string se sastoji iz glavne adrese mehanizma za pretraživanje, za kojim slede neki parametri. Prvi parametar, p, označava reči koje tražimo. Drugi parametar, *n=100*, označava broj sajtova koji se dobija; ne možete proizvoljno koristiti brojeve, već ste ograničeni na nekoliko vrednosti, a 100 je najveća moguća vrednost. Parametar *h=s* označava da program treba da traži web sajtove (ne kategorije), a poslednji parametar, *b=1*, označava broj početnog elementa. Da biste dobili sajtove od 101 do 200, trebalo bi da poslenji parametar zamenite sa *b=101*.

UPOZORENJE

Program WebFind radi sa serverom na Yahoo web sajtu kada je ova knjiga pisana i testirana. Softver na sajtu se može menjati svakog dana, što može sprečiti da se WebFind pravilno izvršava. ■

Metod `Execute` procesa, koji aktivira poziv `Resume`, je sastavljen iz dva dela. U prvom delu program se povezuje sa HTTP serverom pozivanjem funkcije `InternetOpen` i upotrebom rezultujućeg hendla za poziv funkcije `InternetOpenURL`. Drugi poziv daje hendl na URL koji možete proslediti funkciji `InternetReadFile` da biste čitali blokove podataka. Podaci se čuvaju u lokalnom stringu i, dok program dobija podatke, program takođe može da ažurira stausnu liniju glavnog formulara. Kada su svi podaci pročitani, program zatvara vezu ka URL-u i Internet sesiju dva puta pozivajući funkciju `InternetCloseHandle`. Evo prvog dela metoda `Execute`:

```
procedure TFindWebThread.Execute;
var
  hHttpSession, hReqUrl: HInternet;
```

```
  Buffer: array [0..1023] of Char;
  nRead: Cardinal;
  strRead: string;
  begin, nEnd: Integer;
begin
  strRead := '';
  hHttpSession := InternetOpen ('FindWeb',
    INTERNET_OPEN_TYPE_PRECONFIG, nil, nil, 0);
  try
    hReqUrl := InternetOpenURL (hHttpSession, PChar(StrUrl),
      nil, 0,0,0);
    strStatus := 'Connected to ' + StrUrl;
    Synchronize (ShowStatus);
    try
      // read all the data
      repeat
        InternetReadFile (hReqUrl, @Buffer,
          sizeof (Buffer), nRead);
        strRead := strRead + string (Buffer);
        strStatus := 'Retrieved ' + IntToStr (Length (strRead)) +
          ' of ' + StrUrl;
        Synchronize (ShowStatus);
      until nRead = 0;
    finally
      InternetCloseHandle (hReqUrl);
    end;
  finally
    InternetCloseHandle (hHttpSession);
  end;
```

Drugi deo metoda izdvaja URL-ove koji se odnose na web sajtove iz rezultata, koji se nalazi u stringu `strRead`. Program traži naredna pojavljivanja podstringa `href="http`, kopirajući tekst sve do karaktera >. Ukoliko pronađeni string sadrži reč yahoo, smatra se lokalnim linkom i izostavlja se iz rezultata. Ovaj deo koda možete pronaći u kodu programa, a izlaz programa možete videti na slici 20.12. Primetićete da sam ja već dobio rezultate zahteva, ali program trenutno dobija drugu stranu, što je naznačeno na statusnoj liniji. Možete istovremeno započeti više pretraživanja, ali imajte na umu da će se rezultati dodati u istu Memo komponentu.

SLIKA 20.12 *Aplikacija WebFind se može koristiti za pronalaženje liste sajtova iz Yahoo mehanizma za pretraživanje*

Dinamičke web strane

Kada pregledate web sajt, obično preuzimate statičke strane — tekstualne fajlove u HTML formatu — sa web servera na svoj klijent kompjuter. Kao web programer, te strane možete ručno kreirati, ali za većinu poslova mnogo je razumnije izraditi statičke strane na osnovu informacija koje se nalaze u bazi podataka. Ovim pristupom praktično činite snimak podataka, što je prilično razumno ukoliko se podaci ne menjaju veoma često. Ovaj pristup je razmatran u prethodnom poglavlju.

Alternativa statičkim HTML stranama je izrada dinamičkih strana. Da biste ovo učinili, Vi izdvajate informacije direktno iz baze podataka kao odgovor na zahtev pretraživača, tako da HTML koji se šalje Vašoj aplikaciji prikazuje aktuelne podatke, a ne stari zamrznuti pogled na podatke. Ovakav pristup ima smisla ukoliko se podaci često menjaju.

Kao što je ranije pomenuto, postoji nekoliko načina na koje možete programirati ponašanje web servera, a to su idealni načini za dinamičko generisanje HTML strana. Dva najčešća protokola za programiranje web servera su CGI (Common Gateway Interface) i Web server API. Treća tehnika, Active Server Pages — ASP, postaje veoma popularna. Ja ću ASP razmatrati na kraju ovog poglavlja jer Delphi 5 sadrži specifičnu podršku za ASP.

NAPOMENA

Imajte na umu da Delphijeva WebBroker tehnologija (koja je na raspolaganju kako u Enterprise tako i u Professional izdanju Delphija) smanjuje razlike između CGI-ja, WinCGI-ja i ISAPI-ja obezbeđivanjem klase okruženja. Na ovaj način lako možete pretvoriti CGI aplikaciju u WinCGI aplikaciju, ili je unaprediti u ISAPI model. ■

Pregled CGI-ja

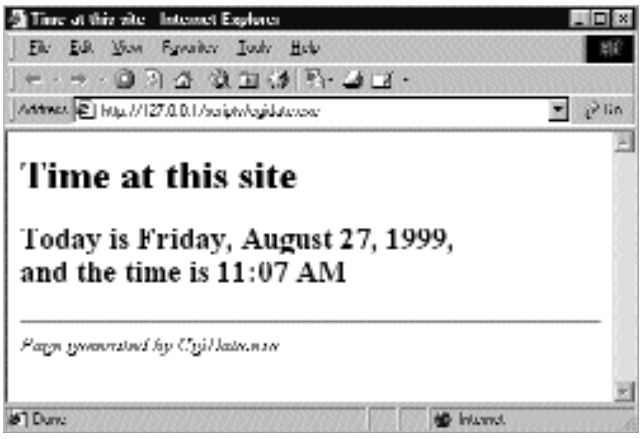
CGI je standardni protokol za komunikaciju između klijent pretraživača i web servera. To nije naročito efikasan protokol, ali je veoma rasprostranjen i ne zavisi od platforme. Ovaj protokol omogućava pretraživaču da zatraži i pošalje podatke, i zasnovan je na standardnom ulazu i izlazu komandne linije aplikacije (obično aplikacije na konzoli). Kada server detektuje zahtev za stranom za CGI aplikaciju, server pokreće aplikaciju, prosleđuje podatke sa komandne linije iz zahteva za stranom u aplikaciju, a zatim šalje standardni izlaz aplikacije nazad u klijent kompjuter.

Postoje mnogi alati i jezici koje možete upotrebiti za pisanje CGI aplikacija, a Delphi je jedan od njih. Sa očiglednim ograničenjem da Vaš web server mora biti Windows NT ili Windows 95 sistem zasnovan na Intelu, možete izraditi dovoljno sofisticirane CGI programe u Delphiju. Uprkos činjenici da se naziva standardom, zapravo postoje različite vrste CGI-ja. Tradicionalni CGI koristi standardni ulaz i izlaz sa komandne linije kao i promenljive okruženja. WinCGI koristi INI fajl koji se prosleđuje kao parametar komandne linije aplikaciji (umesto promenljivih okruženja), i specifične ulazne i izlazne fajlove (umesto ulaza/izlaza komandne linije). Proizvođači servera su WinCGI prevashodno namenili za Visual Basic programere koji ne mogu pristupiti promenljivima okruženja. Još jedna nova varijanta, nazvana FastCGI, je načinjena da učini ceo proces pozivanja CGI aplikacija mnogo bržim, ali još nema široku podršku.

Da biste izradili CGI program, a da ne koristite nijednu klasu podrške, možete jednostavno kreirati Delphi aplikaciju za konzolu, ukloniti tipičan izvorni kod projekta i zameniti ga sledećim iskazima:

```
program CgiDate;
{$APPTYPE CONSOLE}
uses SysUtils;
begin
  writeln ('HTPP/1.0 200 OK');
  writeln ('CONTENT-TYPE: TEXT/HTML');
  writeln;
  writeln ('<HTML><HEAD>');
  writeln ('<TITLE>Time at this site</TITLE>');
  writeln ('</HEAD><BODY>');
  writeln ('<H1>Time at this site</H1>');
  writeln ('<HR>');
  writeln ('<H3>');
  writeln (FormatDateTime(
    '"Today is " dddd, mmmm d, yyyy,' +
      '"<br> and the time is" hh:mm:ss AM/PM',
    Now));
  writeln ('</H3>');
  writeln ('<HR>');
  writeln ('<I>Page generated by CgiDate.exe</I>');
  writeln ('</BODY></HTML>');
end.
```

CGI programi proizvode zaglavlje za kojim sledi HTML tekst koji koristi standardni izlaz. Ukoliko program direktno izvršite, tekst će se prikazati u prozoru terminala. Ukoliko ga umesto toga pokrenete sa web servera, a izlaz pošaljete pretraživaču, pojaviće se formatirani HTML tekst, kao što možete videti na slici 20.13.



SLIKA 20.13 *Izlaz aplikacije CgiDate, onako kako se vidi u Microsoftovom Internet Exploreru*

Izrada naprednih i komplikovanih aplikacija upotrebom običnog CGI-ja zahteva mnogo posla. Na primer, da biste izdvojili informaciju o statusu HTTP zahteva potrebno je da pristupite odgovarajućim promenljivim okruženja:

```
// get the path name
GetEnvironmentVariable ('PATH_INFO',
  PathName, sizeof (PathName));
```

Pregled ISAPI-ja/NSAPI-ja

Potpuno dugačiji pristup je upotreba Web server API-ja, popularnog ISAPI-ja (Internet Server API, koji je predstavio Microsoft) i manje uobičajenog NSAPI-ja (Netscape server API). Ovi API-ji Vam omogućavaju da napišete DLL koji server učitava u svoj adresni prostor i obično zadržava u memoriji neko vreme. Kada učita DLL, server može obraditi pojedinačne zahteve preko procesa koji su deo glavnog procesa, umesto pokretanja novog EXE-a za svaki zahtev kao što to mora učiniti kada su u pitanju CGI aplikacije.

Kada server primi zahtev za stranom, učitava DLL (ukoliko to već nije učinio) i izvršava odgovarajući kod, koji može pokrenuti novi proces ili upotrebiti postojeći za obradu zahteva za stranom (IIS Web server nudi podršku za prozivanje da bi se izbeglo kreiranje novog procesa za svaki zahtev). Kod DLL-a zatim šalje odgovarajuće podatke klijentu koji zahteva stranu. Pošto se ova komunikacija odvija u memoriji, ova vrsta aplikacija je mnogo brža od CGI aplikacija, a posmatrani sistem će na ovaj način moći da podrži više simultanih zahteva za stranama.

Glavni nedostatak server API DLL-ova je da njihova čvrsta veza sa serverom predstavlja Ahilovu petu; ukoliko DLL blokira ili izazove nedostatak memorije, ceo web server će se blokirati. Ipak, najnovije verzije Microsoftovog IIS Web servera rešavaju ovaj problem izvršavanjem DLL-a u *zaštićenom* prostoru. Drugi problem se javlja kada je DLL u memoriji i kada ne možete kompajlirati ažuriranu verziju; potrebno je da iz memorije izbacite DLL ili da zaustavite web server (operaciju koju možete obaviti samo na kompjuteru za testiranje).

ISAPI DLL-ovi se mnogo ne razlikuju od običnih Windows DLL-ova. Ovi DLL-ovi moraju izvesti nekoliko specifičnih funkcija koje će web server pozivati: funkcije `GetExtensionVersion` i `HttpExtensionProc`. Server prvu funkciju poziva kada učitava DLL prvi put, a drugu funkciju za svaki naredni zahtev. Parametri ovih funkcija su složene strukture podataka koje čuvaju ulazne podatke i metode servera koje možete pozivati da biste dobili rezultat. Sledi primer ove funkcije (preuzet iz primera IsapiDemo), koji koristi polje `lpszPathInfo` i funkciju `WriteClient`:

```
function HttpExtensionProc (
  var ECB: TEXTENSION_CONTROL_BLOCK): DWORD; stdcall;
var
  OutStr: string;
  StrLength: Cardinal;
begin
  with ECB do
  begin
    OutStr :=
      '<HTML><HEAD><TITLE>First Isapi Demo</TITLE></HEAD><BODY>' +
      '<H2><CENTER>First Isapi Demo</CENTER></H2>' +
      'Hello Mastering Delphi Readers...<p><hr>' +
      '<b>Activated by ' + PChar (@lpszPathInfo[1]) + '</b><p>' +
```

```
      '<i>From IsapiDLL on ' + DateToStr (Now) +
      ' at ' + TimeToStr (Now) + '</i>' +
      '</body></html>';
    StrLength := Length (OutStr);
    WriteClient(ConnID, PChar (OutStr), StrLength, 0);
  end;
  Result := HSE_STATUS_SUCCESS;
end;
```

NAPOMENA

Program ne koristi jednostavno parametar `lpszPathInfo`, već koristi podstring počevši od drugog karaktera, da bi se oslobodio početnog karaktera /. Budimo precizniji, izraz `PChar (@lpszPatnInfo[1])` uzima string počevši od memorijske adrese drugog karaktera putanje (niz karaktera zasnovanih na nuli). ■

Delphijeva WebBroker tehnologija

CGI i ISAPI fragmenti koda koje sam Vam prikazao do sada, demonstriraju običan, direktan pristup protokolu i API-ju. Proširivanje primera na tom nivou je svakako moguće, ali ono što je interesantno je upotreba takozvane WebBroker tehnologije, specifične hijerarhije klasa u okviru VCL-a koje su ugrađene da bi pojednostavile programiranje za Web na strani servera, i specifični tip modula podataka koji se nazivaju WebModules. I Enterprise i Professional izdanja Delphija 5 sadrže ovo okruženje.

Upotrebom WebBroker tehnologije možete veoma lako započeti programiranje ISAPI ili CGI aplikacija. Na prvoj strani (*New*) Object Repositoryja odaberite ikonu Web Server Aplication. Okvir za dijalog koji sledi će Vam ponuditi tri alternative, ISAPI, CGI i WinCGI, kao što možete videti na slici 20.14. Ukoliko odaberte prvu opciju, Delphi će za Vas generisati osnovnu strukturu ISAPI aplikacije.

SAVET

Za početnu tačku Vaše aplikacije na strani servera takođe možete upotrebiti DB Web Application Wizard, koji se nalazi na strani Business okvira za dijalog File➥New. Ovaj čarobnjak generiše program upotrebom tabele ili upita koji su povezani sa komponentom DataSetTableProducer. Ovo može biti od pomoći, ali kod koji se generiše je prilično ograničen. ■

SLIKA 20.14 *Alternativne opcije za izradu web server aplikacije u Delphi Enterprise izdanju*

Aplikacija koju Delphi generiše (bez obzira na to koji tip odaberete) zasnovana je na klasi `TWebModule`, kontejneru koji je veoma sličan modulu podataka. WebModule kod je sličan kodu modula podataka, što ćemo ubrzo videti, ali vredi pogledati kod biblioteke:

```
library Project1;
uses
  WebBroker, ISAPIApp,
  Unit1 in 'Unit1.pas' {WebModule1: TWebModule};
{$R *.RES}
exports
  GetExtensionVersion,
  HttpExtensionProc,
  TerminateExtension;
begin
  Application.Initialize;
  Application.CreateForm(TWebModule1, WebModule1);
  Application.Run;
end.
```

UPOZORENJE

U Delphiju 5 Vaš kod se mora referisati na novu WebBroker jedinicu. Postojeće WebBroker aplikacije koje se referišu na HTTPApp jedinicu se moraju ažurirati ili nećete moći da ih kompajlirate. Ova izmena je uvedena da bi se smanjile restrikcije koje se odnose na upotrebu paketa samo za vreme izvršavanja za komponente web server aplikacija. ■

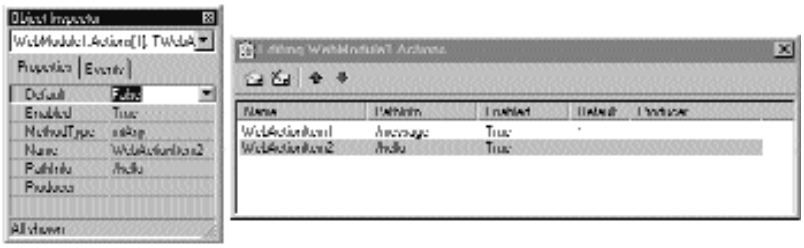
Mada je ovo biblioteka koja izvozi ISAPI funkcije, kod izgleda slično kodu aplikacije. Ipak, u kodu se koristi trik — `Application` objekat koji koristi program nije tipični globalni objekat klase `Tapplication`, već je objekat nove klase. Ovaj novi `Application` objekat je `TISAPIApplication` klase (ili `TCGIAppliaction` klase ukoliko izrađujete taj tip aplikacije), koja je izvedena iz klase `TWebApplication`.

Mada ove klae aplikacije obezbeđuju osnovu, nećete ih često koristiti u opštem slučaju (kao što to ne žinite sa `Application` objektom u Delphi aplikacijama koje koriste formulare). Najvažnija operacija se obavlja u komponenti WebModule. Ova komponenta je izvedena iz klase `TCustomWebDispatcher`, koja obezbeđuje podršku za svaki ulaz i izlaz Vašeg programa.

Zapravo, klasa `TCustomWebDispatcher` definiše svojstva `Request` i `Response`, koja čuvaju zahtev klijenta i odgovor koji nameravamo da pošaljemo klijentu. Svako od ovih svojstava je definisano upotrebom osnovne apstraktne klase (`TWebRequest` i `TWebResponse`), ali aplikacija inicijalizuje upotrebom specijalnog objekta (kakav su, recimo, objekti `TISAPIRequest` i `TISAPIResponse` potklasa). Ove klase čine dostupnim sve informacije koje se prosleđuju serveru, tako da imate jedan, jednostavan pristup svim informacijama. Isto važi i za odgovor, kojim se veoma lako može manipulisati. Jedna prednost nad ovim pristupom je u tome da ISAPI DLL napisan u ovakvom okruženju veoma nalikuje CGI aplikaciji; zapravo, često imaju identične izvrone kodove.

Ukoliko je ovo struktura Delphijevog okruženja, kako napisati kod aplikacije? Dakle, u komponenti WebModule možete upotrebiti Actions editor (prikazan na slici 20.15), za definisanje niza akcija (koje se čuvaju u nizu Actions ovog svojstva) u zavisnosti od naziva putanje (*path name*) zahteva. Ovaj naziv putanje je deo CGI ili ISAPI URL-a aplikacije, koji dolazi posle naziva programa a pre parametara, recimo kao `path1` u narednom URL-u:

```
http://www.website.com/scripts/cgitest.exe/path?param1=date
```



SLIKA 20.15 *Editor svojstava Actions programa WebModule i svojstva jedne od akcija u Object Inspectoru*

Obezbeđivanjem različitih akcija Vaša aplikacija lako može odgovoriti na zahteve sa različitim nazivima putanja i možete dodeliti različite Producer komponente ili pozvati različite obrade `OnAction` događaja za svaki mogući naziv putanje. Naravno, možete izostaviti naziv putanje da biste obradili generički zahtev. Uzmite u obzir to da umesto da degradirate Vašu aplikaciju u WebModule, možete upotrebiti običan modul podataka i dodati mu komponentu WebDispatcher. Ovo je dobar pristup ukoliko želite da postojeću Delphi aplikaciju pretvorite u proširenje web servera. WebModule objedinjuje WebDispatcher i ne zahteva ga kao posebnu komponentu.

UPOZORENJE

Actions komponente WebDispatcher nema nikakve veze sa Actionsom koji se čuva u komponenti `TactionList`. ■

Kada definišete prateće HTML strane koje pokreće aplikacija, linkovi će načiniti zahteve za stranom URL-ova svake od ovih putanja. Postojanje jednog ISAPI DLL-a koji može obaviti različite operacije u zavisnosti od parametra (u ovom slučaju naziva putanje), omogućava serveru da kopiju ovog DLL-a zadrži u memoriji i da mnogo brže reaguje na zahteve korisnika. Isto delimično važi i za CGI aplikacije: server mora da pokrene nekoliko instanci, ali može keširati fajl i tako ga brže učiniti dostupnim.

Događaj `OnAction` je mesto gde smeštate kod da biste naznačili *odgovor* (response) za određeni *zahtev* (request), dva glavna parametra koja se prosleđuju obradi događaja. Evo jednostavnog primera:

```
procedure TWebModule1.WebModule1WebActionItem1Action(
  Sender: TObject; Request: TWebRequest;
  Response: TWebResponse; var Handled: Boolean);
begin
  Response.Content :=
    '<HTML><HEAD><TITLE>Hello Page</TITLE></HEAD><BODY>' +
    '<H1>Hello</H1>' +
    '<hr><I>Page generated by Marco</I>' +
    '</BODY></HTML>';
end;
```

Svojstvo `Content` parametra `Response` je mesto gde unosite HTML kod koji želite da korisnici vide. Jedini nedostatak ovog koda je da će izlaz pretraživača biti korektno prikazan u više linija, ali kada pogledate HTML izvorni kod, videćete jednu liniju koja odgovara celom stringu. Da biste HTML izvorni kod učinili čitljivijim, deleći ga na više linija, možete umetnuti karakter `#13` za novu liniju.

Da biste omogućili drugim akcijama da obrade ovaj zahtev, odredićete vrednost `False` za poslednji parametar, parametar Handled. U suprotnom, unapred određena vrednost je `True`, i kada obradite zahtev Vašom akcijom, WebModule pretpostavljla da ste završili. Veći deo ISAPI koda aplikacije će se nalaziti u obradi događaja `OnAction` za akcije definisane u WebModule kontejneru. Ove akcije primaju zahtev od klijenta i daju odgovor upotrebom parametara `Request` i `Response`.

Kada koristite `Producer` komponente, Vaš događaj `OnAction` često, kao `Response.Content` daje sadržaj `Producer` komponente, upotrebom jednostavnog dodeljivanja. U Delphiju 5 možete skratiti ovaj kod dodeljivanjem komponente `Producer` svojstvu `Producer` same akcije, a da ne morate više pisati ove jednostavne obrade događaja.

Izrada višenamenskog programa WebModule

Da bih demonstrirao koliko lako na strani servera možete izraditi aplikaciju koja je bogata opcijama upotrebom Delphi podrške, ja sam kreirao primer BrokDemo. Ovaj primer se može kompajlirati kao CGI ili ISAPI aplikacija, tako što ćete jednostavno odabrati odgovarajući fajl projekta. WebModule dele dva projekta, čiji izvorni kodovi nemaju razlike, što je praktičan dokaz da upotrebom WbBroker okruženja možete ISAPI prevesti u CGI i obrnuto. U praksi nameravam da programe testiram upotrebom CGI-ja (da bih izbegao zaustavljanje servera da bih oslobodio biblioteku i ponovo je kompajlirao) i da ih zatim prosledim upotrebom ISAPI-ja.

NAPOMENA

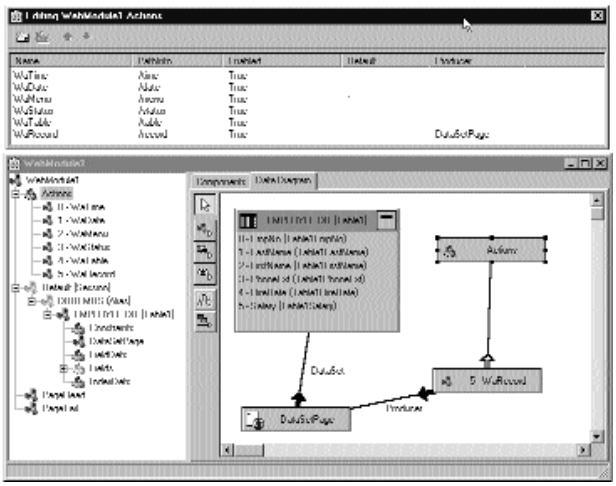
Ukoliko je Vaš cilj da izradite IASPI aplikaciju, takođe možete upotrebiti specifične alate za debagovanje ISAPI DLL-a. Jedan od tih alata, koji je nazvan IntraBob, je izradio Bob Svart (Bob Swart) i dostupan je na web sajtu (`www.drbob42.com`). ■

Ključni element je lista akcija koje ćemo podržati ovom aplikacijom, koje možete videti u Actions editoru na slici 20.16. Akcije su, takođe, vidljive u Designeru web modula podataka, tako da možete videti grafički prikazane njihove zavisnosti sa objektima baze podataka, kao što je prikazano na slici. Ukoliko proučite sliku ili izvorni kod, primetićete da sam ja svakoj akciji dodelio specifični naziv. Takođe sam dodelio sugerišuće nazive obradama događaja `OnAction`. Na primer, `TimeAction` kao naziv metoda je mnogo razumljiviji od naziva `WebModule1WebActionItem1Action` koji Delphi automatski generiše.

Svaka akcija ima drugačiji naziv putanje, od kojih je jedan označen kao unapred određen i izvršava se čak i kada se ne navede naziv putanje. Prva intersantna ideja u ovom programu je upotreba dve PageProducer komponente, koje se koriste za počeni i krajnji odeljak svake strane, `PageHead` i `PageTail`. Centralizovanje ovog koda čini lakšim menjanje koda, naročito ukoliko je baziran na spoljašnjim HTML fajlovima. HTML koji proizvodi ove komponente se dodaje na početak i na kraj rezultujućeg HTML-a i obradi `OnAfterDispatch` događaja web modula:

```
procedure TWebModule1.WebModule1AfterDispatch(
  Sender: TObject; Request: TWebRequest;
  Response: TWebResponse; var Handled: Boolean);
begin
  Response.Content := PageHead.Content +
    Response.Content + PageTail.Content;
end;
```

Ja dodajem početni i krajnji HTML na kraju generisanja strane jer to jednostavno omogućava da komponente proizvedu HTML kao da komponente sastavljaju sav kod. Započinjanje od HTML-a u metodu `OnBeforeDispatch` znači da ne možete direktno da dodelite Producer komponente akcijama, ili će Producer komponente zaobići Content koji ste obezbedili kao sadržaj.



SLIKA 20.16 *Akcije primera BrokDemo, koje su prikazane Actions editorom i Data Module Designerom*

Događaj `OnBeforeDispatch` uzima naziv skripta da bi ga učinio dostupnim događajima PageProducer komponente (koji kao parametar ne prihvataju `Request`). Evo dva fragmenta koda koji pokazuju ovaj kombinovani efekat:

```
procedure TWebModule1.WebModule1BeforeDispatch(
  Sender: TObject; Request: TWebRequest;
  Response: TWebResponse; var Handled: Boolean);
begin
  // code shared by all actions
  ScriptName := Request.ScriptName;
  Table1.Open;
end;
procedure TWebModule1.PageTailHTMLTag(Sender: TObject;
  Tag: TTag; const TagString: String; TagParams: TStrings;
  var ReplaceText: String);
begin
  if TagString = 'script' then
    ReplaceText := ScriptName;
end;
```

Ovaj kod je aktiviran da proširi <#script> tag svojstva HTMLDoc komponente PageTail. Kod akcija datuma i vremena je direktan. Zaista zanimljiv deo počinje od putanje Menu, što je unapred određena akcija. U ovoj obradi događaja OnAction aplikacija jednostavno izrađuje listu raspoloživih akcija, obezbeđujući link za svaku od njih (tagom <a>) u for petlji:

```
procedure TWebModule1.MenuAction(
  Sender: TObject; Request: TWebRequest;
  Response: TWebResponse; var Handled: Boolean);
var
  I: Integer;
begin
  Response.Content := '<H3>Menu</H3><ul>'#13;
  for I := 0 to Actions.Count - 1 do
    Response.Content := Response.Content +
      '<li> <a href="' + ScriptName +
      Action[I].PathInfo + '"> ' + Action[I].Name + '</a>'#13;
  Response.Content := Response.Content + '</ul>';
end;
```

Druga akcija primera BrokDemo obezbeđuje korisnicima listu sistemskih podešavanja koja se odnose na zahtev, nešto što je prilično korisno za debagovanje. Takođe je poučno videti koliko informacija, a ne tačno koje informacije, HTTP protokoli prosleđuju od pretraživača do web servera i obrnuto. Da bi proizveo ovu listu, program traži vrednost za svako svojstvo klase TWebRequest, kao što pokazuje ovaj početni fragment:

```
procedure TWebModule1.StatusAction(
  Sender: TObject; Request: TWebRequest;
  Response: TWebResponse; var Handled: Boolean);
var
  I: Integer;
begin
  Response.Content := '<H3>Status</H3>'#13 +
    'Method: ' + Request.Method + '<br>'#13 +
    'ProtocolVersion: ' + Request.ProtocolVersion + '<br>'#13 +
    'URL: ' + Request.URL + '<br>'#13 +
    'Query: ' + Request.Query + '<br>'#13 + ...
```

Dinamička izrada izveštaja baze podataka

Primer BrokDemo definiše još dve akcije, naznačene /table i /record nazivima putanja. Za ove dve poslednje akcije naš program proizvodi glavnu listu naziva, a zatim prikazuje detalje jednog sloga upotrebom komponente DataSetTableProducer za formatiranje cele tabele i komponente DataSetPageProducer za izradu pogleda sloga. Slede svojstva ovih dveju komponenata:

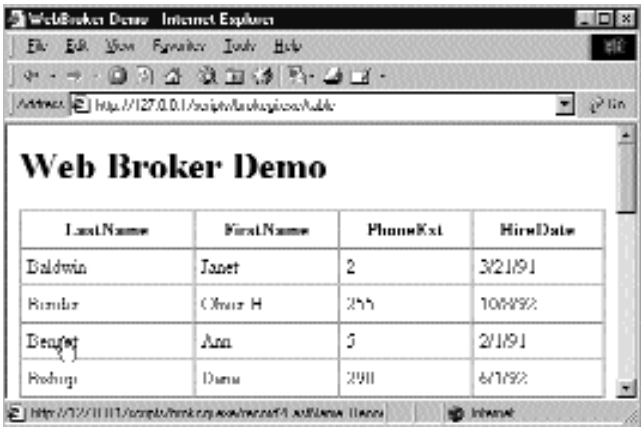
```
object DataSetTableProducer1: TDataSetTableProducer
  DataSet = Table1
  OnFormatCell = DataSetTableProducer1FormatCell
end
object DataSetPage: TDataSetPageProducer
  HTMLDoc.Strings = (
    '<H3>Employee: <#LastName></H3>'
    '<ul><li> Employee ID: <#EmpNo>'
    '<li> Name: <#FirstName> <#LastName>'
```

```
    '<li> Phone: <#PhoneExt>'
    '<li> Hired On: <#HireDate>'
    '<li> Salary: <#Salary></ul>')
  OnHTMLTag = PageTailHTMLTag
  DataSet = Table1
end
```

Da bismo proizveli celu tabelu, jednostavno povezujemo komponentu DataSetTableProducer sa `Producer` svojstvom odgovarajućih akcija, a da ne obezbeđujemo nijednu obradu događaja. Tabela je učinjena moćnijom dodavanjem internih linkova određenim slogovima. Naredni kod se izvršava za svaku ćeliju tabele, ali se aktivira samo za prvu kolonu ili prvi red (onaj koji sadrži zaglavlje):

```
procedure TWebModule1.DataSetTableProducer1FormatCell(
  Sender: TObject; CellRow, CellColumn: Integer;
  var BgColor: THTMLBgColor; var Align: THTMLAlign;
  var VAlign: THTMLVAlign; var CustomAttrs, CellData: String);
begin
  if (CellColumn = 0) and (CellRow <> 0) then
    CellData := '<a href="' + ScriptName + '/record?LastName=' +
      Table1['LastName'] + '&FirstName=' + Table1 ['FirstName'] +
      '"> ' + CellData + ' </a>';
end;
```

Na slici 20.17 možete videti rezultat ove akcije. Kada korisnik selektuje jedan od linkova, program se ponovo poziva, proverava `QueryFields` listu stringova i izdvaja parametre iz URL-a. Zatim koristi vrednosti koje odgovaraju poljima tabele za pretraživanje slogova (koje se zasniva na pozivu `FindNearest`).



SLIKA 20.17 *Izlaz koji odgovara putanji tabele primera BrokDemo, koji proizvodi HTML tabelu sa internim linkovima upotrebom Delphi komponenata*

```
procedure TWebModule1.RecordAction(
  Sender: TObject; Request: TWebRequest;
  Response: TWebResponse; var Handled: Boolean);
begin
  Table1.Open;
```

```
  // go to the requested record
  Table1.FindNearest ([Request.QueryFields.Values['LastName'],
    Request.QueryFields.Values['FirstName']]);
  // get the output
  Response.Content := Response.Content +
    DataSetPage.Content;
end;
```

NAPOMENA

Primer koji smo upravo izradili pristupa Paradox tabeli preko BDE-a. CGI verzija se izvršava po jednom za svaki zahtev i zapravo će učitati i izbaciti BDE svaki put kada se pokrene. Možete razmisliti o upotrebi ISAPI-ja, pristupajući podacima iz lokalnog fajla ili izvršavanjem druge BDE aplikacije na serveru, tako da BDE ostane učitan u memoriji. ■

O upitima i formularima

Prethodni primer je koristio neke HTML Producer komponente koje su predstavljene ranije u ovom poglavlju. Postoji još jedna komponenta ove grupe koju do sada nismo koristili, komponenta QueryTableProducer. Kao što ćemo odmah videti, ova komponenta čini veoma lakim čak i izradu složenih programa za baze podataka. Pretpostavimo da želite da pronađete neke kupce u bazi podataka. Možete načiniti sledeći HTML (ugnežđen u HTML tabelu radi boljeg formatiranja):

```
<h4>Customer QueryProducer Search Form</h4>
<form action="/scripts/CustQueP.dll/search" method="POST">
<table>
<tr><td>State:</td>
  <td><input type="text" name="State"></td></tr>
<tr><td>Country:</td>
  <td><input type="text" name="Country"></td></tr>
<tr><td></td>
  <td><center><input type="Submit"></center></td></tr>
</form>
```

NAPOMENA

Kao u Delphiju, HTML formular sadrži niz kontrola (obično stvari kao što su polja za unos). Postoje vizuelni alati koji Vam pomažu da dizajnirate ove formulare, ili možete ručno uneti odgovarajući HTML kod. Kontrole koje su na raspolaganju su kontrole (buttons), ulazni tekst (ili polja za izmene), selekcije (ili combo polja) i opcione kontrole (ili ulazne kontrole). Kontrole možete definisati kao specifične tipove, recimo, kao Submit ili Reset, koji impliciraju standardno ponašanje. Važan element formulara je metod zahteva (request method), koji može biti POST (podaci se šalju u pozadini, a prihvatate ih u svojstvu `ContentFields`) ili GET (podaci se šalju kao deo URL-a, a izdvajate ih iz svojstva `QueryFields`). ■

Izlaz ovog formulara možete videti na slici 20.18. Postoji još jedan važan element na koji treba obratiti pažnju: nazivi ulaznih komponenata (*State* i *Country*) se moraju podudarati sa parametrima Query komponente:

```
select
  Company, State, Country
from
  CUSTOMER.DB
where
  State = :State or Country = :Country
```

Ovaj kod se koristi u primeru CustQueP (Customer Query Producer). Da bih ga izradio, smestio sam komponentu Query unutar WebModula i za njega sam generisao objekte polja. U isti WebModule sam dodao komponentu QueryTableProducer povezanu sa `Producer` svojstvom `/search` akcije. ISAPI DLL će generisati odgovarajući odgovor. Kako ovo funkcioniše? Kada aktiviramo komponentu QueryTableProducer pozivanjem njene funkcije `Content`, ona inicijalizuje Query komponentu dobijanjem parametara iz HTTP zahteva. Komponenta može automatski proveriti metod zahteva i zatim upotrebiti svojstvo `QueryFields` (ukoliko je zahtev GET) ili svojstvo `ContentFields` (ukoliko je zahtev POST).

SLIKA 20.18 *HTML formular koji se koristi u primeru CustQueP, a koji je formatiran postavljanjem kontrola u HTML tabelu*

Jedan problem koji se javlja prilikom upotrebe statičkog HTML formulara koji smo i ranije načinili je da nam on ne govori koje države može pronaći za nas. Možemo upotrebiti kontrolu selekcije umesto kontrole za editovanje u HTML formularu. Ipak, ukoliko korisnik doda nove slogove tabeli baze podataka, potrebno je da automatski ažuriramo listu elemenata. Kao konačno rešenje možemo dizajnirati ISAPI DLL da bismo tokom rada proizveli formular, a kontrole za selekciju možemo popuniti raspoloživim elementima.

HTML za ovu akciju ćemo generisati `/form` akcijom, koju smo povezali sa komponentom PAgeProducer. PageProducer sadrži sledeći HTML tekst, koji ugnežđuje dva specijalna taga:

```
<h4>Customer QueryProducer search Form</h4>
<form action="CustQueP.dll/search" method="POST">
<table>
<tr><td>State:</td><td><select name="State">
<#state>
</select>
<tr><td>Country:</td><td><inputname="Country">
<option></option>
<#Country>
</select>
</td></tr>
<tr><td></td><center><input type="Submit"></center></td></tr>
</form>
```

Primetićete da tagovi imaju iste nazive kao i neka polja tabele. Kada PageProducer naiđe na neki od ovih tagova, dodaje <option> HTML tag za svaku pojedinačnu vrednost odgovarajućeg polja. Sledi kod obrade događaja OnTag, koji je prilično generički i može se iznova koristiti:

```
procedure TWebModule1.PageProducer1HTMLTag(
  Sender: TObject; Tag: TTag; const TagString: String;
  TagParams: TStrings; var ReplaceText: String);
begin
  ReplaceText := '';
  Query2.SQL.Clear;
  Query2.SQL.Add ('select distinct ' +
    TagString + ' from customer');
  try
    Query2.Open;
    try
      Query2.First;
      while not Query2.EOF do
      begin
        ReplaceText := ReplaceText +
          '<option>' + Query2.Fields[0].AsString +
          '</option>'#13;
        Query2.Next;
      end;
    finally
      Query2.Close;
    end;
  except
    ReplaceText := '{wrong field: ' + TagString + '}';
  end;
end;
```

Ovaj metod koristi drugu Query komponentu, koju sam ručno smestio na formular i povezao sa DBDemos bazom podataka, i proizvodi izlaz prikazan na slici 20.19.

Konačno, ovo web server proširenje, kao mnoga druga koja smo izradili, omogućava korisniku da pogleda detalje o određenom slogu. Kao u prethodnom primeru, ovo možemo postići prilagođavanjem izlaza prve kolone (kolone nula), koja je generisana komponentom QueryTableProducer:

```
procedure TWebModule1.QueryTableProducer1FormatCell(
  Sender: TObject; CellRow, CellColumn: Integer;
  var BgColor: THTMLBgColor; var Align: THTMLAlign;
  var VAlign: THTMLVAlign; var CustomAttrs, CellData: String);
begin
  if (CellColumn = 0) and (CellRow <> 0) then
    CellData := '<a HREF="' + Request.ScriptName +
      '/record?' + CellData + '">' + CellData + '</a>'#13;
end;
```

SLIKA 20.19 *Akcija formulara primera CustQueP proizvodi HTML formular sa komponentom za selekciju koja je ažurirana da bi odslikala trenutni status baze podataka*

Akcija za ovaj link je /record i prosledi će određeni element posle parametra ? (bez naziva parametra, što je, pomalo, van standarda). Kod koji koristimo za izradu HTML tabela za slogove ne koristi Producer komponente kao što smo to mi činili; umesto toga, kod je veoma sličan kodu jednog od prvih ISAPI primera:

```
procedure TWebModule1.RecordAction(
  Sender: TObject; Request: TWebRequest;
  Response: TWebResponse; var Handled: Boolean);
var
  I: Integer;
begin
  if Request.QueryFields.Count = 0 then
    Response.Content := 'Record not found'
  else
  begin
    Query2.SQL.Clear;
    Query2.SQL.Add ('select * from customer ' +
      'where Company="' + Request.QueryFields[0] + '" ');
    Query2.Open;
    Response.Content :=
      '<HTML><HEAD><TITLE>Customer Record</TITLE></HEAD><BODY>'#13 +
      '<H1>Customer Record: ' + Request.QueryFields[0] +
      '</H1>'#13 +
      '<table border>'#13;
    for I := 1 to Query2.FieldCount - 1 do
      Response.Content := Response.Content +
        '<tr><td>' + Query2.Fields [I].FieldName +
        '</td>'#13'<td>' + Query2.Fields [I].AsString +
        '</td></tr>'#13;
    Response.Content := Response.Content +
      '</table><hr>'#13 +
```

```
      // pointer to the query form
      '<a HREF="' + Request.ScriptName + '/form">' +
      ' Next Query </a>'#13 +
      '</BODY></HTML>'#13;
  end;
end;
```

Brojač poseta web strane

Aplikacije na strani servera koje smo do sada izradili su bile zasnovane samo na tekstu. Naravno, lako možete dodati reference na postojeće grafičke fajlove. Ono što je još interesantnije je izrada programa na strani servera koji mogu generisati grafike koje se menjaju tokom vremena.

Tipičan primer je brojač poseta strane. Da biste napisali web brojač, trenutni broj poseta čuvamo u fajlu, a zatim čitamo i uvećavamo vrednost svaki put kada se program brojača pozove. Kako ćemo prikazati ovu informaciju? Ukoliko nam je potreban samo HTML tekst sa brojem poseta, kod je prilično direktan:

```
procedure TWebModule1.WebModule1WebActionItem1Action(
  Sender: TObject; Request: TWebRequest;
  Response: TWebResponse; var Handled: Boolean);
var
  nHit: Integer;
  LogFile: Text;
  LogFileName: string;
begin
  LogFileName := 'WebCont.log';
  System.Assign (LogFile, LogFileName);
  try
    // read if the file exists
    if FileExists (LogFileName) then
    begin
      Reset (LogFile);
      Readln (LogFile, nHit);
      Inc (nHit);
    end
    else
      nHit := 0;
    // saves the new data
    Rewrite (LogFile);
    Writeln (LogFile, nHit);
  finally
    Close (LogFile);
  end;
  Response.Content := IntToStr (nHit);
end;
```

Ono što je malo interesantnije je izrada grafičkog brojača koji se lako može umetnuti na bilo koju HTML stranu. U osnovi postoje dva pristupa za izradu grafičkog brojača: možete pripremiti bitmapu za svaku cifru i zatim ih kombinovati u programu, ili jednostavno možete prepustiti programu da iscrta memorijsku bitmapu da bi proizveo grafiku koju želite da prikažete. U programu WebCount sam odabrao drugi pristup.

U osnovi, možemo kreirati Image komponentu koja čuva memorijsku bitmapu, koja se može iscrtati upotrebom uobičajenih metoda klase TCanvas. Zatim možemo pridružiti bitmapu TJpegImage objektu. Pristupanje bitmapi preko komponente JpegImape konvertuje sliku u JPEG format. U ovom trenutku možemo sačuvati JPEG podatke na tok i dati ih kao rezultat. Kao što možete videti, postoji dosta koraka, ali kod zapravo nije složen:

```
// create a bitmap in memory
Bitmap := TBitmap.Create;
try
  Bitmap.Width := 120;
  Bitmap.Height := 25;
  // draw the digits
  Bitmap.Canvas.Font.Name := 'Arial';
  Bitmap.Canvas.Font.Size := 14;
  Bitmap.Canvas.Font.Color := RGB (255, 127, 0);
  Bitmap.Canvas.Font.Style := [fsBold];
  Bitmap.Canvas.TextOut (1, 1, 'Hits: ' +
    FormatFloat ('###,###,###', Int (nHit)));
  // convert to JPEG and output
  Jpeg1 := TJpegImage.Create;
  try
    Jpeg1.CompressionQuality := 50;
    Jpeg1.Assign(Bitmap);
    Stream := TMemoryStream.Create;
    Jpeg1.SaveToStream (Stream);
    Stream.Position := 0;
    Response.ContentStream := Stream;
    Response.ContentType := 'image/jpeg';
    Response.SendResponse;
    // the response object will free the stream
  finally
    Jpeg1.Free;
  end;
finally
  Bitmap.Free;
end;
```

Tri iskaza koja su odgovorna za slanje JPEG slike su dva iskaza koja određuju svojstva ContentStream i ContentType za Response i poslednji poziv SendResponse. Tip sadržaja se mora podudarati sa jednim od mogućih MIME tipova koje prihvata pretraživač, a redosled ova tri iskaza je važan. Postoji i metod SendStream objekta Response, ali ga treba pozvati samo posle slanja tipa podataka u okviru zasebnog poziva.

Efekat ovog programa možete videti na slici 20.20. Da bih to postigao, dodao sam sledeći kod HTML strani:

```
<img src="http://127.0.0.1/scripts/webcount.exe"
  border=0 alt="hit counter">
```

SLIKA 20.20 *Grafički brojač poseta web strane*

Obrada informacija o pošti

U poslednjem primeru ću Vam pokazati kako da upotrebite aplikaciju na strani servera za generisanje poruke sa specijalnim formatiranjem; ove poruke će biti obrađivane korisničkim programom. Zašto generisati poruke umesto da podatke lokalno sačuvamo na kompjuteru servera? Protokoli elektronske pošte se mogu koristiti za moćne i složene transakcije između dva korisnika koja nisu stalno povezana na Internet. U ovom slučaju upotreba programa koji direktno koriste priključke ne funkcioniše. Serveri pošte nude način za desinhronizaciju server i klijent aplikacija.

Drugim rečima, udaljeni korisnik zna kada se dogodila aktivnost na sajtu, ili kada je neko upotrebio program koji je generisao elektronsku poštu, jednostavnom proverom naloga za poštu. Ukoliko imate dodatni nalog za poštu (a danas ga je prilično lako dobiti), takođe možete automatizovati proces verifikacije pisanjem programa koji automatski izdvaja i obrađuje poruke.

CGI server pošte

Da bih ilustrovao obe strane veze elektronske pošte, napisao sam dva jednostavna programa. Prvi je CGI aplikacija na strani servera koja koristi SMTP FastNet komponentu (koja je opisana ranije u ovom poglavlju). Sledi DFM fajl za module podataka:

```
object WebModule1: TWebModule1
  Actions = <
    item
      Default = True
      Name = 'WebActionItem1'
      OnAction = WebModule1WebActionItem1Action
    end>
  object Mail: TNMSMTP
    Host = 'XXX'
    Port = 25
    ReportLevel = 0
    UserID = 'marco'
    PostMessage.ToAddress.Strings = (
      'marco@AST')
    PostMessage.Body.Strings = (
      'Subscription')
```

```
    PostMessage.Subject = 'Subscribe'
  end
end
```

Naravno, potrebno je da ažurirate program odgovarajućim adresama za SMTP host, odgovarajućim `UserID-om` i adresom elektronske pošte gde želite da šaljete poruke, `PostMessage.ToAddress`. Program sadrži samo jedan metod, obradu svoje jedine akcije. Ovaj metod izdvaja informacije koje korisnik mora uneti u odgovarajući HTML formular, šalje poruku i prikazuje rezultujući poruku korisniku:

```
procedure TWebModule1.WebModule1WebActionItem1Action(
  Sender: TObject; Request: TWebRequest;
  Response: TWebResponse; var Handled: Boolean);
var
  OutString: string;
begin
  OutString := Request.ContentFields.Values ['firstname'];
  OutString := OutString + ' ' +
    Request.ContentFields.Values ['lastname'];
  OutString := OutString + ' [' +
    Request.ContentFields.Values ['email'] + ']';
  // send email
  Mail.PostMessage.FromAddress := OutString;
  Mail.Connect;
  Mail.SendMail;
  Mail.Disconnect;
  Response.Content := Response.Content +
    '<HTML><HEAD><TITLE>Newsletter</TITLE></HEAD>' +
    '<BODY><H1>Newsletter</H1><H2>Subscription received</H2><hr>' +
    '<H4>You''re registered in our database as <br>' +
    OutString + '</h4>' +
    '</BODY></HTML>';
end;
```

HTML formular koji koristi program je prikazan na slici 20.21, a njegov HTML izvorni kod je izlistan ispod slike. HTML kod je interesantan iz dva razloga. Prvo, polja za izmene HTML formulara koja se koriste za unos sadrže name, koje koristi CGI aplikacija za dobijanje ulaznih podataka.

SLIKA 20.21 *HTML formular za unos programa WebMail*

Drugo, koristi jednostavan skript napisan u jeziku JavaScript (pogledajte `OnSubmit` odeljak formulara) da proveri da li su polja za izmene prazna pre nego što se pošalje zahtev. Sledi HTML kod:

```
<HTML><HEAD>
<TITLE>Subscription</TITLE>
</HEAD>

<BODY  bgcolor='#FFFFFF'>
<H1>Subscription Module</H1>

Fill the following form to subscribe to my newsletter.

<form
name= "subscribe"
action="/cgi-bin/WebMail1.exe/new"
method= "post"
onSubmit="
if(!subscribe.lastname.value ¦¦
!subscribe.firstname.value ¦¦
!subscribe.email.value)
    {
alert( 'All fields must be filled');
return false;
    };">
<TABLE>
<TR>
<TD><B>Fi rst Name:</B></TD>
<TD><input name="firstname">
</TD>
</TR>
<TR>
<TD><B>Last Name : </B></TD>
<TD><input name="lastname"></TD>
</TR>
<TR>
<TD><B>Email: </B></TD>
<TD><input name="email"></TD>
</TR>
<TR>
<TD></TD>
<TD><input type=submit value="Send"></TD>
  </TR>
<TABLE>
</form>

</BODY></HTML>
```

Ukoliko poznajete C++, verovatno Vam je poznat i jezik JavaScript, jer koristi sličnu sintaksu. Ja ovde ne želim da detaljno razmatram JavaScript, ali sam želeo da Vam pokažem da klijent strana programa za Web može biti učinjena još moćnijom uvođenjem skriptova i upotrebom drugih karakteristika HTML-a. Deo koji se vidi je izrađen u HTML formularu i nije naročito fleksibilan kako Windows aplikacije mogu biti, ali HTML sa nešto skripta nudi sve osnovne karakteristike koje su potrebne za pristojan formular za unos.

Dobijanje zahteva na osnovu poruka

Druga strana aplikacije je program koji se koristi za dobijanje poruka koje generiše CGI server proširenje. Ovaj program je nazvan GetMail i nalazi se u istom direktorijumu kao i program WebMail. Program GetMail je zasnovan na formularu i sadrži komponentu NMPOP3. Moraćete da ažurirate program odgovarajućim nazivom `Host`, `UserID-om` i `Passwordom`.

Formular takođe sadrži listu i dve kontrole, koje se koriste za premeštanje svih novih prijavljenih korisnika u listu i njihovo čuvanje u fajlu. Memo komponenta se koristi za prikazivanje grešaka i poruka dnevnika. Program se povezije na POP3 server, čita broj poruka, a zatim skenira svaku poruku u obrnutom redosledu. Pozivanje metoda `GetMailMessage` popunjava MailMessage svojstvo komponente. U tom trenutku, program proverava da li je u pitanju poruka prijave (proverom polja `Subject`), izdvaja pošiljaoca, dodaje njegovo ime i adresu elektronske pošte listi i uklanja poruku sa servera. Bilo koja poruka koja ima drugačije zaglavlje se ne uklanja; umesto toga, tekst poruke se dodaje Memo komponenti.

Active Server Pages

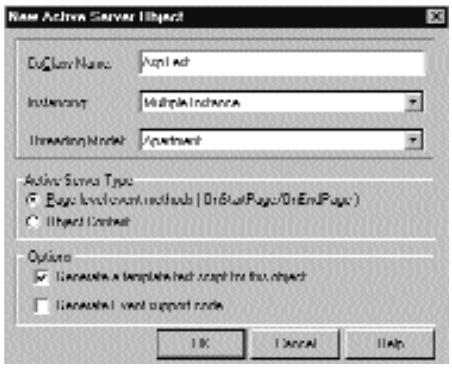
Jedan od najvećih problema ISAPI i CGI aplikacija je činjenica da slede pravila HTTP protokola, koji nema proveru. Svaki zahtev koji pristiže od bilo kog korisnika se smatra potpuno novim zahtevom. Postoje mnoge tehnike koje možete upotrebiti da biste rešili ovaj problem, uključujući i upotrebu "kolačića" (cookies) — što je prilično jednostavno upotrebom WebBroker arhitekture — i upotrebu sakrivenih polja formulara koji prosleđuju ID korisnika sa strane na stranu.

Drugo rešenje je upotreba nove Microsoftove tehnologije, Active Server Pages (ASP — aktivne strane servera). Ideja koja stoji iza ASP-a je dodavanje skripta HTML kodu, tako da je deo teksta web strane direktno dostupan dok se ostale informacije mogu dodati u vreme izvršavanja na serveru. Klijent dobija običan HTML fajl. Razlika između ovog pristupa i ISAPI-ja je da nije potrebno da ponovo kompajlirate program na serveru da biste videli izmene; potrebno je samo da ažurirate skript. ASP nudi složen model, gde možete pridružiti stalne podatke sesiji (na primer, korisniku koji se pomera sa strane na stranu odeljka Vašeg web sajta) i celoj aplikaciji (odeljku web sajta, bez obzira na korisnika).

ASP je prilično složena tehnologija i ovde ću je razmatrati samo u odnosu na Delphi programiranje. Jedna od karakteristika ASP-a je da Vam omogućava da kreirate COM objekte u okviru skripta, a te COM objekte možete napisati u Delphiju. Delphi 5 čak obezbeđuje sepcifične klase za podršku i čarobnjaka koji Vam pomaže da izradite ASP objekte. U poređenju sa ISAPI-jem ili CGI-jem, jedna od prednosti je da Vaši ASP objekti izrađeni u Delphiju mogu pristupiti sesiji i informacijama aplikacije, baš kao što čini ASP skript. To znači da možemo automatski dobiti dodatne karakteristike kao stalne podatke ugrađene u objekat na strani servera. Izradom kompajliranog ASP objekta takođe možemo povećati brzinu složenog koda na strani servera. (ASP skriptovi nisu najbolje rešenje kada se gorvori o performansama.) Ali, ja ne želim da detaljno razmatram ASP, već ću obratiti pažnju samo na Delphi podršku.

Da biste ovo isprobali, jednostavno kreirajte novu ActiveX biblioteku, a zatim pokrenite Active Server Object Wizard (sa ActiveX strane okvira za dijalog File➥New). Kao što možete videti na slici 20.22, čarobnjak sadrži nekoliko opcija. Objekat koji je integrisan sa ASP skriptom možete izraditi selektovanjem opcije Page-Level Event Metods, ili možete izraditi interni objekat (koji se

može instalirati kao MTS objekat) upotrebom opcije Object Context. Samo u prvom slučaju objekat automatski obrađuje metod `OnStartPage`, koji kao parametar prihvata *kontekst skripta* (scripting context). U oba slučaja, VCL klase iz kojih vršite izvođenje (klase `TASPObject` i `TASPMTSObject`, repektivno) sadrže svojstva za pritup `Request`, `Response`, `Session`, `server` i `Application` ASP objektima.



SLIKA 20.22 *Novi Active Server Object Wizard*

Kada ste kreirali ASP objekat uz pomoć čarobnjaka (ja sam koristio opciju Page-Level Event Metods za primer AspTest), Delphi će prikazati editor tipa biblioteke gde možete pripremiti listu svojstava i metoda za Vaš ASP objekat. Jednostavno dodajte karakteristike koje su Vam potrebne, a zatim napišite njihov kod. Na primer, možete napisati sledeći jednostavan test metod:

```
procedure Tasptest.ShowData;
begin
  Response.Write ('<h3>Delphi wrote this text</h3>');
end;
```

i aktivirati ga iz narednog ASP skripta (koji je samo malo izmenjen u odnosu na demo skript koji za Vas generiše Delphi):

```
<h4>Message</h4>
<% Set DelphiASPObj = Server.CreateObject("asptest1.asptest")
   DelphiASPObj.showData
%>
```

Interesantan element je da isti skript (ili neki drugi ASP skript iste aplikacije) takođe može odrediti globalne vrednosti kojima naš Delphi objekat može pristupiti. Slično, više objekata može komunicirati, određivati globalne promenljive aplikacije i promenljive sesije za određenog korisnika. Na primer, možemo dodati sledeći tekst ASP strani:

```
<h4>hello</h4>
<%
   Session.Value("UserName") = "Marco"
   DelphiASPObj.Hello
 %>
```

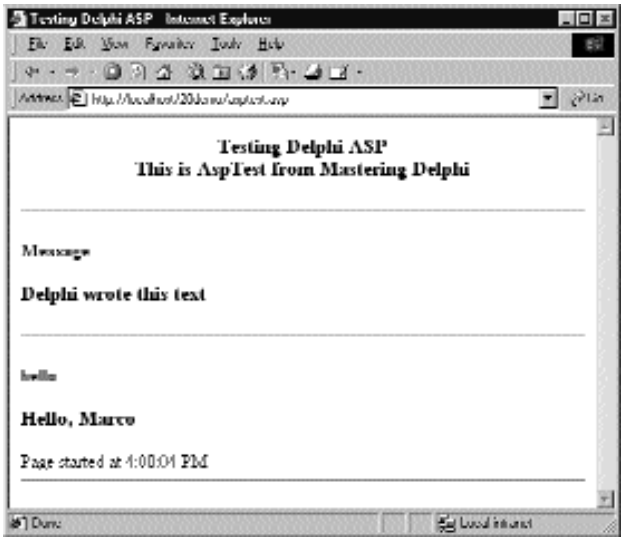
Ja sam napisao kod koji se koristi za određivanje svojstva i metod zahteva jedan za drugim, ali se oni mogu nalaziti i na različitim stranama. Ovo novo *dinamičko* svojstvo (Microsoftov termin za ove vrednosti koje se dodaju objektu) se čuva u sesiji, tako da zavisi od trenutnog korisnika. Metod `Hello` može upotrebiti korisničko ime za pozdravnu poruku:

```
procedure Tasptest.Hello;
var
  strName: string;
begin
  strName := Session ['UserName'];
  Response.Write ('<h3>Hello, ' + strName + '</h3>');
  Response.Write ('<p>Page started at ' + TimeToStr (StartTime);
end;
```

Rezultat ovog i prethodnog koda možete videti na slici 20.23. Poslednja linija metoda koristi promenljivu koju sam odredio kada je strana prvi put učitana, u metodu `OnStartPage` (uprkos nazivu ovo nije obrada događaja, već metod koji će ASP mehanizam pozvati kada se aktivira strana koja sadrži objekat):

```
procedure Tasptest.OnStartPage(const AScriptingContext: IUnknown);
begin
  inherited OnStartPage(AScriptingContext);
  StartTime := Now;
end;
```

Ovaj metod dobija kontekst skripta. Osnovna klasa `TASPObejct` koristi metod za inicijalizaciju svih ASP objekata (uključujući dva objekta `Response` i `Session` koje koristim u kodu), prikazujući ih kao svojstva.



SLIKA 20.23 *Web strana generisana AspTest objektom koji sam izradio pomoću Delphija*

```
procedure Tasptest.OnStartPage(const AScriptingContext: IUnknown);
begin
  inherited OnStartPage(AScriptingContext);
  StartTime := Now;
end;
```

Ovaj metod dobija kontekst skripta. Osnovna klasa `TASPObejct` koristi metod za inicijalizaciju svih ASP objekata (uključujući dva objekta `Response` i `Session` koje koristim u kodu), prikazujući ih kao svojstva.

Da biste generisali složeniji HTML iz Delphi ASP objekta, možete upotrebiti Producer komponente, opciono ih povezujući sa skupom podataka. U primeru AspTest dodao sam komponentu Table i komponentu DataSetTableProducer, povezao sam ih na uobičajen način i napisao sam sledeći kod da bih ih aktivirao:

```
procedure Tasptest.ShowTable;
begin
  DataModule1 := TDataModule1.Create (nil);
  try
    Response.Write (DataModule1.DataSetTableProducer1.Content)
  finally
    DataModule1.Free;
  end;
end;
```

Više smisla će imati da kreirate modul podataka kada se kreira i ukloni COM objekat (zaobilazeći `Initialize` i `Destroy`) ili kada se strana učita i izbaci iz memorije (upotrebljavajući `OnStartPage` i `OnEndPage`).

Šta je sledeće?

U ovom dugom poglavlju sam Vam predstavio tehnike programiranja koje se odnose na Internet: generisanje HTML koda, upotrebu ActiveX kontrola i ActiveForma na web stranama, veze niskog nivoa priključaka, neke Internet protokole visokog nivoa, CGI i ISAPI tehnologije na strani servera, WebBroker okruženje i ASP.

Tehnologije Internet programiranja dobijaju sve više pažnje, ali su i veoma nestabilne i neozbiljne, a mnoge alternative često dovode do istih rezultata. Zbog toga sam pokušao da Vam dam veoma širok prikaz raspoloživih tehnologija, primenjujući ih na niz raznovrsnih primera. O ovoj temi ćete više naučiti u narednom poglavlju gde se razmatra Internet Express arhitektura, koju Delphi 5 dodaje MIDAS tehnologiji aplikacija distribuiranih baza podataka.

POGLAVLJE

21

Paralelne (Multitier) aplikacije za baze podataka

Velike kompanije često imaju veće potrebe nego što aplikacije koje koriste lokalne baze podataka i SQL serveri mogu pružiti. U proteklih nekoliko godina Borland je počeo da odgovara potrebama velikih korporacija, pa je čak promenio i svoje ime u Inprise da bi podvukao fokus novih projekata. Postoje mnoge nove tehnologije koje Delphi podržava: višelinijska (three-rier) arhitektura zasnovana na Windowsu NT i DCOM-u, CORBA arhitektura zasnovana na NT i UNIX serverima, TCP/IP i aplikacije priključaka i — najviše od svega — baze podataka za Web. Poglavlje 20 je pokazalo kako da napišete aplikacije distribuiranih baza podataka upotrebom priključaka. Ovo poglavlje će predstaviti ključne ideje Delphijeve podrške višelinijske strukture, upotrebom MIDAS, DCOM, TCP/IP, MTS, CORBA tehnologija i novom Internet Express tehnologijom. Ja ću obratiti više pažnje na aspekte programiranja ovih arhitektura nego na instalaciju i konfigurisanje (ovi aspekti se razlikuju na različitim operativnim sistemima i previše su komplikovani da bi se detaljno objasnili). Ovo poglavlje treba da bude samo uvod u neke složene tehnologije koje se odnose na Delphi programiranje (za koje bi inače bila potrebna posebna knjiga da bi se opisale), kako je to potrebno, recimo, za CORBA tehnologiju.

Pre nego što nastavim, potrebno je da naglasim dva važna elementa. Prvo, alati koji podržavaju ovakvu vrstu programiranja su na raspolaganju samo u Enterprise izdanju Delphija; drugo, u nekim slučajevima ćete morati da platite Borlandu da biste mogli da prosledite neophodan softver na strani servera, što je slučaj sa MIDAS tehnologijom. Ovaj drugi zahtev čini ovu arhitekturu isplativom samo kada su u pitanju veliki sistemi (to jest, serveri koji su povezani sa velikim brojem klijenata). Plaćanje licence je neophodno samo za prosleđivanje server aplikacije. Može se platiti manja cena za server (bez obzira na broj klijenata koji će biti priključeni) ili se može platiti po klijentu ukoliko planirate da povežete samo nekoliko njih. Plaćanje licence nije potrebno kada je u pitanju programiranje i evaluacija.

NAPOMENA

Potrošićete novac na MIDAS licencu, ali novac možete uštedeti na SQL server klijent licencama. Kompanije su sačuvale desetine hiljada dolara na godišnjim SQL server licencama, priključujući stotine ili hiljade kompjutera na MIDAS server umesto na SQL server. MIDAS serveru je potrebna samo jedna SQL server licenca, a krajnjim korisnicima nije potrebna nijedna. ■

Jedan, dva, tri nivoa

U početku su PC aplikacije za baze podataka bile rešenja samo za klijente: program i fajlovi baze podataka su se nalazili na istom kompjuteru. Kasnije su programeri vizionari prebacili fajlove baze podataka na mrežni server fajlova. Klijent kompjuteri su i dalje sadržali softver aplikacije i ceo mehanizam baze podataka, ali fajlovima baze podataka sada može da pristupa više korisnika istovremeno. Još uvek možete da koristite ovakvu konfiguraciju upotrebom Delphija i Paradox fajlova (ili, naravno, samog Paradoxa), ali ovaj pristup je bio rasprostranjen pre samo nekoliko godina.

Sledeći veliki pomak je bilo klijent/server programiranje, koje je Delphi podržavao od svoje prve verzije. U klijent/server svetu, klijent kompjuter zahteva neke podatke od server kompjutera, koji sadrži kako fajlove baze podataka tako i mehanizam baze podataka pomoću koga se pristupa podacima. Ovakva arhitektura smanjuje značaj uloge klijenta, ali smanjuje i njegove potrebe za moćnom obradom na klijent kompjuteru. U zavisnosti od toga kako programeri implementiraju klijent/server, server može obaviti veći deo obrade podataka (ako ne i svu). Na ovaj način, moćan server može da obezbedi obradu podataka za nekoliko manje moćnih klijenata.

Prirodno, postoje i mnogi drugi razlozi za upotrebu centralizovanih servera podataka, kakvi su zaštita podataka i integritet, jednostavnije strategije za pravljenje rezervnih kopija, centralno upravljanje podacima i tako dalje. Server baze podataka se često naziva SQL server, jer je to jezik koji se najčešće koristi za pravljenje upita nad podacima, ali se naziva i DBMS (DataBase Management System — sistem upravljanja bazom podataka), što odslikava činjenicu da serveri obezbeđuju alate za manipulisanje podacima, kao što je podrška pravljenju rezervnih kopija i replikacija.

Naravno, neke aplikacije koje izrađujete možda ne moraju u potpunosti da iskoriste prednosti DBMS-a, tako da jednostavno rešenje na strani klijenta može biti dovoljno. S druge strane, možda će Vam biti potrebna robusnost DBMS sistema, ali na jednom, izolovanom kompjuteru. U ovom slučaju možete upotrebiti lokalnu verziju SQL servera, kakav je Local InterBase (koji je deo Professional i Enterprise izdanja Delphija). Tradicionalno klijent/server programiranje se obavlja višelinijskom arhitekturom. Ipak, ukoliko DBMS primarno obavlja čuvanje podataka umesto

obrade podataka, klijent može da sadrži oba koda korisničkog interfejsa (formatiranje korisničkog ulaza i izlaza upotrebom izveštaja, formulare za unos podataka, upite i tako dalje) i kod koji se odnosi na manipulisanje podacima (koji je poznat i kao poslovna — *business* — *pravila*). U ovom slučaju, dobra je ideja da pokušate da razdvojite ova dva dela programa i izgradite lokalnu višelinijsku arhitekturu. Termin *logička* ovde označava da i dalje postoje dva kompjutera (to jest, dve fizičke linije), ali smo sada podelili aplikaciju na tri jasno razdvojena elementa.

Delphi 2 je predstavio podršku za logičku višelinijsku arhitekturu upotrebom modula poataka. Kao što se sećate, modul podataka je nevizuelni kontejner za komponente aplikacije za pristupanje podacima, ali često sadrži nekoliko obrada događaja koji se odnose na baze podataka. Možete da podelite jedan modul podataka između nekoliko različitih formulara i obezbedite različite korisničke interfejse za iste podatke; može postojati jedan ili više formulara za unos podataka, više izveštaja, master/detail formulara i različtitih formulara sa grafikom i dinamičkim izlazom.

Logički višelinijski pristup rešava mnoge probleme, ali takođe ima i nekoliko nedostataka. Prvo, morate da replicirate deo programa koji se odnosi na manipulisanje podacima na različite klijent kompjutere, što može da umanji performanse, ali je mnogo veći problem složenost održavanja koda. Drugo, kada više klijenata menja iste podatke, ne postoji jednostavan način za obradu rezultujućih konflikata ažuriranja. I, na kraju, za logičke višelinijske Delphi aplikacije morate da instalirate i konfigurišete BDE na svakom klijent kompjuteru.

Sledeći logički korak od klijent/servera je bio da se deo aplikacije sa modulom podataka pomeri na drugi server kompjuter i da se prilagodi tako da ga koriste svi klijent programi sa kojima komunicira. Ovo je prava namena modula podataka, koji su predstavljeni u Delphiju 3. Udaljeni moduli podataka se izvršavaju na server kompjuteru — koji se generalno naziva server aplikacije. Server aplikacije odmah komunicira sa DBMS-om (koji se može izvršavati na serveru aplikacije ili na drugom kompjuteru). Zbog toga se klijent kompjuteri ne povezuju direktno na SQL server, već indirektno preko servera aplikacije.

U ovom trenutku se postavlja pitanje da li je još uvek potrebno da instaliramo BDE. Tradicionalna Delphijeva klijent/server arhitektura (čak i kada je višelinijska) zahteva da instalirate DBE na svakom od klijenata, a to je operacija koja je problematična kada morate konfigurisati i održavati stotine kompjutera. U novoj fizičkoj višelinijskoj arhitekturi potrebno je da instalirate i konfigurišete BDE samo na serveru aplikacije, a ne na klijent kompjuterima. To znači instaliranje BDE-a i konfigurisanje drajvera i alijasa samo na jednom kompjuteru! Pošto klijent programi imaju samo kod korisničkog interfejsa i veoma ih je lako instalirati, sada ih možemo smestiti u kategoriju takozvanih *lakih klijenata* (thin clients). Marketinškim rečnikom kazano, arhitekturu možemo nazvati arhitekturom lakih klijenata bez konfigurisanja (*non-configuration thin-client architecture*). Ali, posvetimo se tehničkim detaljima a ne marketinškoj terminologiji.

Tehničke osnove: MIDAS

Osnove Delphijeve višelinijske arhitekture čini MIDAS (Middle-tier Distributed Application Services), kolekcija različitih tehnologija koja zajednički funkcioniše radi lakše izrade distribuiranih aplikacija upotrebom Delphija. Delphi sadrži treću verziju ove tehnologije, MIDAS 3, koja je takođe na raspolaganju u C++ Builderu i JBuilderu da bi se omogućili distribuirani projekti za više platformi i za više jezika.

MIDAS je tehnologija koja se primenjuje na strani servera, te ćete morati da ga instalirate na međukompjuteru, kompjuteru koji klijent kompjuterima obezbeđuje podatke dobijene sa SQL servera baze podataka ili drugih izvora podataka. Bilo da se međuserver aplikacije i SQL server nalaze na različitim kompjuterima, bilo da se nalaze na istom kompjuteru, to nije od važnosti niti utiče na MIDAS arhitekturu.

Slično, MIDAS ne zahteva SQL server za čuvanje podataka. MIDAS može da prosleđuje podatke iz različitih izvora, uključujući SQL, CORBA i druge MIDAS servere, ili podatke koji se izračunavaju tokom rada.

Kao što očekujete, klijent strana MIDAS-a je neverovatno mala i lako se prosleđuje. Jedini fajl koji Vam je potreban je nazvan Midas.dll (u prethodnim verzijama je bio nazvan DbClient.dll), mali (260 KB) DLL koji implementira `ClientDataSet` i `RemoteServer` komponente i obezbeđuje vezu sa serverom aplikacije. Ovaj DLL je u osnovi mali, samostalan mehanizam baze podataka. Ovaj DLL kešira podatke iz udaljenog modula podataka i primenjuje pravila koja zahteva Constraint Broker. U MIDAS-u 3 više nije potrebno da registrujete ovaj DLL kao COM server.

Server aplikacije koristi isti DLL za obradu skupova podataka (koji se nazivaju delte — *deltas*) koji se dobijaju od klijenata kada pošalju nove ili ažurirane slogove. Ipak, server zahteva i nekoliko drugih biblioteka koje sve instalira MIDAS.

IAppServer interfejs

U prethodnim verzijama MIDAS-a dve strane aplikacije su komunicirale upotrebom `IDataBroker` i `IProvider` interfejsa. U Delphiju 5 ova dva interfejsa ne postoje, a umesto njih se koristi novi `IAppServer` interfejs. (Na ovaj način je ograničena kompatibilnost sa postojećim programima koji koriste MIDAS 3. Delphi help fajl sadrži informacije o izmenama koje je potrebno učiniti u aplikaciji da bi odgovarala novoj arhitekturi.)

`IAppServer` interfejs zamenjuje karakteristike `IProvider` interfejsa i predstavlja novu važnu karakteristiku: namenjen je za upotrebu sa objektima bez stanja. Upotrebom `IProvider` interfejsa server je čuvao informacije o statusu klijent programa — na primer, koji su slogovi već prosleđeni klijentu. Ovim je bilo otežano prilagoditi obejkte servera tako da odgovaraju slojevima veze bez stanja, kao što je CORBA red poruka i MTS, a takođe i približavanje HTTP i web podrški.

Drugi razlog za prelazak na ovu novu arhitekturu je to što je sada sistem više dinamički (provajderi se sada izvoze određivanjem svojstva, ne promenom biblioteke tipa) i zbog smanjivanja broja poziva (round-trips) što može uticati na performanse. Sada MIDAS čini manje poziva, ali svaki put predaje više podataka.

`IAppServer` interfejs sadrži sledeće metode:

```
AS_ApplyUpdates
AS_GetRecords
AS_DataRequest
AS_GetProviderNames
AS_GetParams
AS_RowRequest
AS_Execute.
```

Često će biti potrebno da ih direktno pozivate, jer postoje Delphi komponente kako na strani klijenta tako i na strani servera koje sadrže ove pozive, čineći ih lakšim (a ponekad ih potpuno sakrivajući). U praksi, u biblioteci tipa udaljenog modula podataka ćete nasleđivati sopstvene interfejse iz ove biblioteke, verovatno dodajući nove metode. U prethodnim verzijama je bilo neophodno izvoziti određene provajder interfejse iz biblioteke tipa servera; sada to nije potrebno, te je potrebno manje prilagođavanja izvedenog interfejsa.

Protokoli povezivanja

MIDAS definiše samo arhitekturu višeg nivoa i može koristiti različite tehnologije za prebacivanje podataka sa srednje linije na stranu klijenta. MIDAS podržava većinu vodećih standarda, uključujući sledeće:

DCOM (ILI DISTRIBUTED COM) Ova tehnologija je direktno dostupna pod Windowsom NT i Windowsom 98 i ne zahteva dodatne aplikacije na serveru. Još uvek je potrebno instalirati je na mašinama sa Windowsom 95. DCOM je u osnovi proširenje COM tehnologije (koju smo razmatrali u poglavljima 15 i 16) koja omogućava klijent aplikaciji da upotrebi objekat servera i izvrši ga na zasebnom kompjuteru.

MTS DCOM takođe omogućava upotrebu MTS-a (Microsoft Transaction Server), koji obezbeđuje karakteristike kao što su zaštita, manipulisanje komponentama i transakcije baza podataka. MTS je na raspolaganju za verzije Windowsa NT i Windowsa 98.

TCP/IP PRIKLJUČCI Ovi priključci su na raspolaganju na većini sistema. Upotrebom TCP/IP-a možete distribuirati klijente preko Weba, gde se DCOM ne može uzeti zdravo za gotovo. Da bi upotrebio priključke, server mora da pokrene ScktSrvr.exe aplikaciju koju obezbeđuje Borland, program koji se može izvršavati bilo kao aplikacija bilo kao servis. Ovaj program prima zahteve korisnika, a zatim ih prosleđuje udaljenom modulu podataka (koji se izvršavaju na istom serveru) upotrebom COM-a. Priključci ne pružaju nikakvu zaštitu od grešaka na strani klijenta, jer se server ne obaveštava i može se desiti da ne oslobodi resurse kada se klijent nepredviđeno isključi.

HTTP Upotreba HTTP-a kao protokola transporta preko Interneta pojednostavljuje povezivanje preko firewallova i proxy servera (kojima u opštem slučaju ne prijaju korisnički TCP/IP priključci). Potrebna Vam je specifična web server aplikacija, httpsrvr.dll, koja prihvata korisničke zahteve i kreira odgovarajuće udaljene module podataka upotrebom COM-a. Ovakve web veze mogu koristiti SSL zaštiti, ali se moraju registrovati dodavanjem poziva `EnableWebTransport` funkcije u metodu `UpdateRegistry`. Na kraju, web veze koje se zasnivaju na HTTP transportu mogu koristiti novu podršku za povlačenje objekata.

NAPOMENA

MIDAS HTTP transport može koristiti XML za format paketa podataka, omogućavajući bilo kojoj platformi ili alatu koji može čitati XML da učestvuje u MIDAS transportu podataka. Ovo je proširenje osnovnog MIDAS formata paketa podataka, koji takođe ne zavisi od platforme. ■

CORBA (COMMON OBJECT REQUEST BROKER ARCHITECTURE) Ova arhitektura je zvanični standard za manipulisanje objektima koji se mogu naći na većini operativnih sistema. U poređenju sa DCOM arhitekturom, prednost je da Vaše klijent i server aplikacije mogu biti napisane upotrebom Java jezika ili drugih proizvoda. Inprise implementacija CORBA arhitekture,

Visigenicov Visibroker ORB, je na raspolaganju u okviru Delphi Enterprise izdanja. CORBA donosi mnoge pogodnosti, uključujući transparentnost pozicije, balansiranje učitavanja i zaštitu od pada ORB softvera.

OLENTERPRISE Još uvek možete koristiti Borlandovu OLEnterprise tehnologiju. Morate instalirati izvršnu verziju OLEnterprise kako na klijent tako i server kompjuterima. Ova tehnologija povezivanja se može koristiti za vezu sa Inprise AppServer i Entera tehnologijama. Ovo nije uobičajeno rešenje, a odgovarajuće komponente za povezivanje nisu više na raspolaganju na Delphi Component Paletti.

Obezbeđivanje paketa podataka

Cela Delphijeva višelinijska arhitektura pristupa podacima je zasnovana na ideji *paketa podataka* (data packets). U ovom kontekstu, paket podataka je blok podataka koji se premešta od servera aplikacije do klijenta ili od klijenta nazad serveru. Paket podataka je podskup skupa podataka. Paket podataka opisuje podatke koje sadrži (obično nekoliko slogova podataka) i spisak naziva i tipova polja podataka. Što je još važnije, paketi podataka sadrže veze, to jest, pravila koja se primenjuju nad skupom podataka. Ove veze ćete obično odrediti na serveru aplikacije, a server ih šalje klijent aplikacijama zajedno sa podacima.

Sva komunikacija između servera i klijenta se odvija razmenom paketa podataka. Provajder na serveru upravlja prenošenjem nekoliko paketa podataka u okviru velikog skupa podataka, a cilj mu je da brže odgovori korisniku. Kada klijent prihvati paket podataka, korisnik može izmeniti slogove koje sadrži. Kao što je ranije pomenuto, tokom ovog procesa klijent takođe prihvata i proverava veze. Kada je klijent ažurirao slogove i poslao nazad paket podataka, paket se naziva delta (delta). Delta paket prati razlike između originalnih i ažuriranih slogova, zapisujući sve izmene koje klijent zahteva od servera. Kada klijent zatraži prihvatanje izmena na serveru, šalje delta paket serveru, a server pokušava da primeni svaku od izmena. Kažem *pokušava*, jer ukoliko je server povezan sa nekoliko klijenata, podaci mogu već biti izmenjeni.

Pošto delta paketi sadrže i originalne podatke, server lako određuje da li je neki drugi klijent promenio podatke. Ukoliko jeste, server inicira događaj `OnReconcileError`, koji je jedan od vitalnih elemenata lakih klijent aplikacija. Drugim rečima, višelinijska arhitektura koristi mehanizam ažuriranja koji je sličan mehanizmu koji Delphi koristi za keširana ažuriranja. `ClientDataSet` upravlja podacima u memoriji, u vrsti keša podataka, i obično čita samo podskup slogova koji su na raspolaganju na strani servera, učitavajući još elemenata samo kada su potrebni. Kada klijent ažurira slogove ili unese nove slogove, ove izmene koje čekaju, čuvaju se u drugom lokalnom kešu na klijentu, *delta kešu* (delta cache).

Klijent, takođe, može da sačuva pakete podataka na disku, što znači da korisnici mogu da rade i kada nisu na vezi. Čak se i informacije o grešci i drugi podaci šalju upotrebom protokola paketa podataka, te je ovo zaista jedan od osnovnih elemenata ove arhitekture.

NAPOMENA

Važno je zapamtiti da paketi podataka ne zavise od protokla. Paketi podataka su samo niz bajtova, te svuda gde možete poslati niz bajtova, možete bajtove proslediti kao paket podataka. Ovo je učinjeno da bi se ova arhitektura učinila pogodnom za više različitih protokola transporta, kao što su DCOM, CORBA, HTTP i TCP/IP priključci. ■

Delphijeva podrška komponenata (na strani klijenta)

Sada kada smo proučili osnove nove višelinijske arhitekture, možemo se usredsrediti na Delphi komponente koje je podržavaju. Za programiranje klijent aplikacija Delphi obezbeđuje ClientDataSet komponente, koje imaju sve standardne mogućnosti vezane za skupove podataka (jer su izvedene iz `TDataSet` klase) ali ne zahtevaju BDE, baš kao i ADO i InterBase Express komponente. U ovom slučaju podaci se prenose preko udaljene veze.

Ova veza sa serverom aplikacije se ostvaruje preko jedne druge komponente koja Vam je takođe neophodna u klijent aplikaciji. Potrebno je da upotrebite jednu od četiri specifične komponente za povezivanje (koje su na raspolaganju na strani Midas):

- Komponenta DCOMConnection se može koristiti na klijent strani za povezivanje sa DCOM i MTS serverom, koji se nalaze na istom kompjuteru ili na nekom drugom kompjuteru koji je naznačen svojstvom `ComputerName`. Veza se ostvaruje sa registrovanim objektom koji je zadat kao `ServerGUID` ili `ServerName`.
- Komponenta CorbaConnection se može koristiti za povezivanje sa CORBA serverom. Potrebno je da naznačite `HostName` (naziv ili IP adresu) da biste naznačili kompjuter servera, `RepositoryID` da biste zahtevali određeni modul podataka koji se nalazi na serveru, i opciono upotrebite svojstvo `ObjectName` ukoliko modul podataka izvozi više objekata.
- Komponenta SocketConnection se može upotrebiti za povezivanje sa serverom preko TCP/IP priključka. Potrebno je da naznačite IP adresu i naziv, i GUID server objekta (u svojstvu `InterceptGUID`). U Delphiju 5 ova komponenta za povezivanje sadrži još jedno svojstvo, `SupportCallbacks`, koje možete isključiti ukoliko ne koristite povratne pozive i želite da prosledite programe za Windows 95 kompjutere koji nemaju instaliran Winsock 2.
- Komponenta WebConnection se koristi za upravljanje HTTP vezom koja se lako može ostvariti preko firewalla. Potrebno je da naznačite URL adresu kopije httpsrvr.dll i naziv ili GUID udaljenog objekta na serveru.

Delphijeva podrška komponenata (na strani servera)

Na strani servera (ili zapravo u sredini) potrebno je da upotrebite udaljeni modul podataka, specijalnu verziju `TDataModule` klase, ili neke od specijalizovanih udaljenih modula podataka za MTS ili CORBA. U Delphiju postoje specijalizovani čarobnjaci za kreiranje svakog od ovih modula podataka.

Jedina specifična komponenta koja Vam je potrebna na strani servera je komponenta DataSetProvider. Potrebna Vam je po jedna od ovih komponenata za svaku tabelu ili upit servera koji želite da bude dostupan na strani klijenta. Komponenta DataSetProvider zamenjuje samostalnu Provider komponentu i interni Provider objekat, koji je u prošlim verzijama ugrađen u potklasu `TBdeDataSet`. Klijent aplikacije će tada koristiti zasebnu komponentu `ClientDataSet` za svaki izvezeni skup podataka (ili provajder) koji žele da koriste.

Izrada primera aplikacije

Sada smo spremni da izradimo primer programa. To će nam omogućiti da posmatramo kako funkcionišu neke od komponenata koje sam upravo opisao, a omogućiće nam i da obratimo pažnju na neke druge probleme, što će baciti svetlost na ostale delove Delphi višelinijske zagonetke. Ja ću izradu delova klijenta i servera aplikacije višelinijske aplikacije podeliti u dva koraka. Prvi korak će biti samo provera tehnologije upotrebom minimalnog broja elemenata. Ti programi će biti veoma jednostavni.

Počevši odatle dodaćemo više mogućnosti klijentu i serveru aplikacije. U svakom od primera prikazaćemo podatke iz lokalnih Paradox tabela i odredićemo sve što je potrebno da Vam omogućimo da testirate programe na zasebnim računarima. Ja neću objašnjavati korake koji su potrebni za instaliranje primera na više kompjutera upotrebom različtih tehnologija — to bi trebalo da bude tema bar jedne knjige. Ja ću Vam samo dati kratak uvod u MTS i CORBA tehnologije kasnije u ovom poglavlju.

Prva aplikacija servera

Server stranu našeg osnovnog primera je veoma lako izraditi. Jednostavno kreirajte novu aplikaciju i dodajte udaljeni modul podataka koristeći odgovarajuću ikonu sa strane Multitier Object Repositoryja. Jednostavni Remote Dataset Wizard će Vam zatražiti naziv klase i stil instanciranja. Kada unesete naziv klase, npr. AppServerOne, i kliknete kontrolu OK, Delphi će dodati modul podataka programu. Ovaj modul podataka će sadržati uobičajena svojstva i metode, ali će njegova klasa imati sledeću deklaraciju:

```
type
  TAppServerOne = class(TRemoteDataModule, IAppServerOne)
  private
    { Private declarations }
  protected
    class procedure UpdateRegistry(Register: Boolean;
      const ClassID, ProgID: string); override;
  public
    { Public declarations }
  end;
```

Pored nasleđivanja iz osnovne klase `TRemoteDataModule` (promena u odnosu na prethodne verzije) ova klasa implementira novi interfejs, `IAppserverOne`, koji se izvodi iz osnovnog Borlandovog interfejsa (`IAppServer`). Klasa takođe zaobilazi unapred određeni metod `UpdateRegistry` za dodavanje podrške priključcima i web transportu, kao što možete videti iz koda koji je generisao čarobnjak. Na kraju jedinice pronaći ćete deklaraciju radionice klase:

```
initialization
  TComponentFactory.Create(ComServer, TAppServerOne,
    Class_AppServerOne, ciMultiInstance, tmApartment);
end.
```

Sada možete dodati Table komponentu modulu podataka, povezati je sa bazom podataka i tabelom i na kraju dodati komponentu DataSetProvider i povezati je. Dobićete jednostavan DFM fajl nalik sledećem fajlu:

```
object AppServerOne: TAppServerOne
  object Table1: TTable
    DatabaseName = 'DBDEMOS'
    TableName = 'employee.db'
  end
  object DataSetProvider1: TDataSetProvider
    DataSet = Table1
    Constraints = True
  end
end
```

Šta je sa glavnim formularom ovog programa? Pa, formular je gotovo nepotreban, te možemo dodati oznaku koja označava da je to formular aplikacije servera. Kada ste izradili server, potrebno je da ga kompajlirate i pokrenete jednom. Ova operacija će ga automatski registrovati kao Automation server na Vašem sistemu, što će ga učiniti dostupnim klijent aplikacijama. Naravno, potrebno je da registrujete server na kompjuteru na kojem želite da ga izvršavate, bilo da je to klijent ili neki kompjuter između.

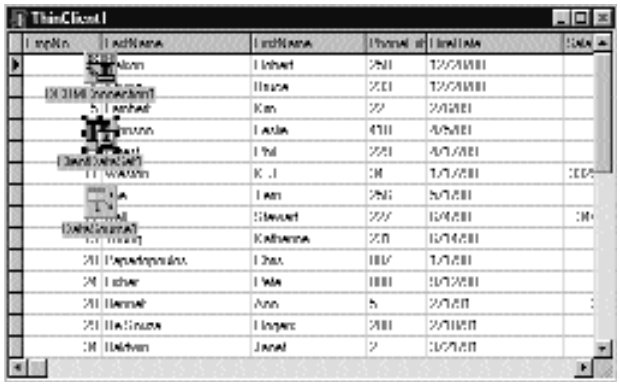
Prvi laki klijent

Sada kada imamo server koji funkcioniše, možemo izraditi klijenta koga ćemo povezati sa serverom. Započećemo proces od standardne Delphi aplikacije i dodaćemo komponentu DCOMConnection (ili odgovarajuću komponentu za određeni tip povezivanja koji želite da testirate). Ova komponenta definiše svojstvo `ComputerName` koje ćete upotrebiti da biste označili kompjuter na kojem se izvršava aplikacija servera. Ukoliko želite da testirate klijenta i aplikaciju servera na istom kompjuteru, svojstvo ćete ostaviti prazno.

Kada ste odabrali kompjuter aplikacije servera, možete jednostavno prikazati listu combo polja svojstva `ServerName` da biste pogledali koji serveri su na raspolaganju, registrovane nazive servera (što je po definiciji naziv izvršnog fajla servera za kojim sledi naziv klase udaljenog modula podataka). Alternativno, možete uneti GUID server objekta u svojstvo `ServerGUID`. Delphi će automatski popuniti ovo svojstvo kada odredite svojstvo `ServerName`, ukoliko može da odredi GUID pretraživanjem Registryja.

U ovom trenutku, ukoliko ste odredili vrednost `True` za svojstvo `Connected` komponente DCOMConnection, prikazaće se formular servera, što je znak da je klijent aktivirao server. Obično nije potrebno da obavite ovu operaciju jer komponenta ClientDataSet aktivira za Vas komponentu Remoteserver. Ja sam ovo učinio samo da bih naglasio šta se dešava u pozadini.

Kao što možda očekujete, sledeći korak je dodavanje komponente ClientDataSet formularu. Morate povezati komponentu ClientDataSet sa komponentom `DCOMConnection1` upotrebom svojstva `RemoteServer`, te stoga i sa jednim od provajdera koji se izvozi. Listu provajdera koje možete upotrebiti možete da vidite u svojstvu `ProviderName` upotrebom combo polja. U ovom primeru moći ćete da odaberete samo `DataSetProvider1` jer je to jedini provajder koji je na raspolaganju za odabrani server. Ova operacija povezuje skup podataka koji se nalazi u memoriji klijenta sa skupom podataka na osnovu fajla koji se nalazi na serveru. Ukoliko aktivirate skup podataka klijenta i dodate nekoliko komponenata koje prepoznaju podatke (ili DbGrid), videćete da ove komponente odmah prikazuju podatke sa servera, kao što je ilustrovano slikom 21.1.



SLIKA 21.1 *Kada aktivirate komponentu ClientDataSet koja je povezana sa udaljenim modulom podataka u vreme dizajniranja, podaci sa servera će postati vidljivi*

Sledi DFM fajl naše minimalne klijent aplikacije, aplikacije ThinCli1:

```
object Form1: TForm1
  Caption = 'ThinClient1'
  object DBGrid1: TDBGrid
    Align = alClient
    DataSource = DataSource1
  end
  object DCOMConnection1: TDCOMConnection
    ServerGUID = '{09E11D63-4A55-11D3-B9F1-00000100A27B}'
    ServerName = 'AppServ1.AppServerOne'
  end
  object ClientDataSet1: TClientDataSet
    Aggregates = <>
    Params = <>
    ProviderName = 'DataSetProvider1'
    RemoteServer = DCOMConnection1
  end
  object DataSource1: TDataSource
    DataSet = ClientDataSet1
  end
end
```

Očigledno, naše prve klijent i server aplikacije su veoma jednostavne, ali pokazuju koliko je lako kreirati pregled skupa podataka koji deli posao na dva izvršna fajla. U ovom trenutku naš klijent služi samo za prikazivanje podataka. Ukoliko na klijentu menjate podatke, time nećete ažurirati fajlove servera. Da biste mogli da ažurirate fajlove servera, potrebno je da dodate još koda klijentu. Pre nego što to učinimo, dodajmo još neke karakteristike serveru.

Dodavanje veza serveru

Kada u Delphiju pišete tradicionalan modul podataka, lako mu možete dodati nešto od logike aplikacije, ili pravila posla, obradom događaja skupa podataka, i određivanjem svojstava polja objekta i obradom njihovih događaja. Ovakav rad bi trebalo da izbegavate na klijent aplikaciji; umesto toga, Vaša pravila posla napišite za središnji čvor.

U prvoj verziji MIDAS arhitekture niste mogli da koristite događaje tabele i polja u središnjem čvoru jer ona nisu bila aktivirana. Alternativno, središnji čvor je mogao i još uvek može poslati neke veze klijentu i može omogućiti klijent programu da te veze primeni tokom unosa korisnika. Počevši od Delphija 4, komponenta DataSetProvider Vam omogućava da pošaljete svojstva polja klijentu (recimo, kao vrednosti min i max), a takođe može obraditi ažuriranja preko skupa podataka koji se koristi za pristupanje podacima (ili upotrebom pratećeg objekta UpdateSql).

Veze polja i tabele

Kada interfejs provajdera kreira pakete podataka koje šalje klijentu, on uključuje definicije polja, veze tabele i polja, i jedan ili više slogova (kako to zahteva komponenta ClientDataSet). To znači da možete prilagoditi središnji čvor i izraditi distribuiranu logiku aplikacije upotrebom veza na osnovu SQL-a.

Veze koje kreirate upotrebom SQL izraza se mogu dodeliti celoj tabeli ili određenim poljima. Provajder šalje veze klijentu zajedno sa podacima, a klijent primenjuje veze pre nego što pošalje nazad ažurirane podatke serveru. Ovim se smanjuje mrežni saobraćaj u poređenju sa situacijom kada imate klijenta koji šalje ažurirane podatke nazad serveru aplikacije i SQL serveru, samo da bi saznao da podaci nisu dobri. Druga prednost kodiranja veza na server strani je u tome da ukoliko se pravila promene, potrebno je da ažurirate samo aplikaciju servera, ali ne i klijenata. Constraint Broker obezbeđuje osnovnu strukturu za ovu logiku.

Ali, kako pišete veze? Postoji nekoliko svojstava koje možete upotrebiti:

- Komponenta Table sadrži svojstvo `Constraints`, koje ima editor svojstva. Slično, svojstvo `Constraints` komponenata BDEDataSet sadrži kolekciju `TCheckConstraint` objekata. Svaki objekat sadrži nekoliko svojstava, uključujući izraz i poruku o grešci.
- Komponenta Query definiše isto svojstvo `Constraint`, a još i svojstvo `Constrained` Boolean.
- Objekat Field definiše svojstva `CustomConstraint`, `ImportedConstraint` i `ConstraintErrorMessage`, koja su funkcionalni ekvivalent svojstava objekta `TCheckConstraint` .

SAVET

Nešto o čemu vredi razmisliti je to da ukoliko koristite rečnik podataka, tada veze možete izdvojiti direktno iz njih. ■

Naš naredni primer dodaje nekoliko veza udaljenom modulu podataka koji je povezan sa tabelom Country.DB baze podataka DBDEMOS. Posle povezivanja tabele sa bazom podataka i kreiranja objekata polja, možete odrediti sledeća specijalna svojstva (veze nad celom tabelom i veze nad pojedinim poljima):

```
object Table1: TTable
  Constraints = <
    item
      CustomConstraint = 'Name <> '''
      ErrorMessage = 'Must provide a name'
      FromDictionary = False
    end>
  TableName = 'COUNTRY.DB'
  object Table1Population: TFloatField
    CustomConstraint = 'Value > 10000'
    ConstraintErrorMessage = 'Population out of range'
    FieldName = 'Population'
  end
end
```

Uključivanje svojstava polja

Možete kontrolisati da li se svojstva objekata polja središnjeg čvora šalju komponenti ClientDataSet (i kopiraju u odgovarajuće objekte polja na strani klijenta) upotrebom vrednosti `poIncFieldProps` svojstva Options komponente DataSetProvider. Ova zastavica kontroliše preuzimanje svojstava polja `Alignment`, `DisplayLabel`, `DisplayWidth`, `Visible`, `DisplayFormat`, `EditFormat`, `MaxValue`, `MinValue`, `Currency`, `EditMask` i `DisplayValues`, ukoliko su na raspolaganju u polju.

Upotrebom ovih vrednosti jednostavno možete napisati Vaš središnji čvor na uobičajen način, izbegavajući upotrebu veza. Ovaj pristup takođe čini lakšim prebacivanje postojećih klijent/server aplikacija u višelinijsku arhitekturu. Glavni nedostatak slanja polja klijentu je da je potrebno vreme za slanje svih dodatnih informacija. Isključivanje vrednosti `poIncFieldProps` dramatično popravlja mrežne performanse skupa podataka koji sadrži veliki broj kolona.

Pored upotrebe upita, server može izdvojiti polja koja se vraćaju klijentu; upit to čini deklarisanjem stalnih objekata polja upotrebom Fields editora i izostavljanjem nekih polja. Pošto polje koje izdvajate može biti potrebno za identifikovanje sloga za naredna ažuriranja (ukoliko je polje deo primarnog ključa), takođe možete upotrebiti svojstvo `ProviderFlags` polja na serveru da biste poslali vrednost polja klijentu, ali da ne bude dostupno komponenti ClientDataSet (ovom se obezbeđuje dodatna zaštita u poređenju sa slanjem polja klijentu i njegovim skrivanjem na klijentu).

Događaji polja i tabele

Možete napisati skup podataka središnjeg čvora i obrade događaja polja kao i obično i omogućiti skupu podataka da obradi primljena ažuriranja na klijentu na tradicionalan način. To znači da se ažuriranja smatraju operacijama nad skupom podataka, identično kao kada korisnik lokalno direktno menja, umeće ili uklanja polja.

Ovo se postiže određivanjem svojstva `ResolveToDataSet` komponente `TDatasetProvider`, povezivanjem bilo skupa podataka koji se koristi za ulaz, bilo skupa koji se korsti za ažuriranje. Zapamtite da ova komponenta provajdera nije ograničena na BDE, već se može koristiti sa bilo kojim skupom podataka. Ovo Vam, takođe, omogućava da uklonite BDE iz središnjeg čvora, što je naročito korisna tehnika ukoliko je središnji čvor deo web servera, gde Vam, možda, nije dozvoljeno da instalirate BDE.

Bilo kojom drugom tehnikom, ažuriranja obavlja skup podataka, što implicira veliku kontrolu (to je Pascal kod!), ali u opštem slučaju slabije performanse. Fleksibilnost je mnogo veća, jer možete koristiti standarno kodiranje. Takođe, prebacivanje postojećih lokalnih ili klijent/server aplikacija za baze podataka, koje koriste skup podataka i događaje polja, je mnogo direktnije kada je u pitanju ovaj model. Ipak, imajte na umu da će korisnik klijent programa primiti poruku o grešci samo kada se lokalni keš (delta) šalje nazad središnjem čvoru. Malo je neobično reći korisniku da podaci koje je pripremio pre otprilike pola sata nisu dobri. Ukoliko dozvolite ovakav pristup, verovatno će biti potrebno da promene u kešu primenite posle svakog `AfterPost` događaja na strani klijenta.

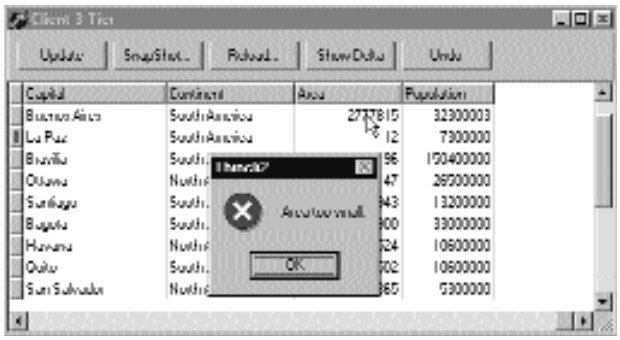
Konačno, ukoliko odaberete ovakvu arhitekturu, Delphi Vam umnogome pomaže u obradi izuzetaka. Bilo koji izuzetak koji se desi na središnjem čvoru događaja ažuriranja (na primer, `OnBeforePost`) automatski se transformiše u Delphi grešku ažuriranja, koja aktivira događaj `OnReconcileError` na strani klijenta (više o događaju kasnije u ovom poglavlju). Nikakav izuzetak se ne prikazuje na središnjem čvoru, ali greška putuje nazad ka klijentu.

Dodavanje karakteristika klijentu

Posle dodavanja veza serveru vreme je da obratimo pažnju na klijent aplikaciju. Prva verzija je bila veoma jednostavna, ali sada postoje brojne karakteristike koje moramo dodati da bi dobro funkcionisala.

Počećemo demonstriranjem toga kako funkcioniše klijent. Da bismo to učinili, proverićemo status sloga i pristupićemo delta informaciji (ažuriranjima koja se šalju nazad serveru). Zatim ćemo programu dodati karakteristike kojima se obrađuju ažuriranja, greške i podrška modelu Briefcase.

Imajte na umu da kada koristite ovog klijenta za lokalne izmene podataka, bićete upozoreni na bilo kakvu grešku u odnosu na pravila rada aplikacije, što je određeno upotrebom veza na strani servera. Server će, takođe, za nas obezbediti unapred određenu vrednost za `Continet` polje novog sloga. Na slici 21.2 možete videti jednu od poruka o grešci koje može prikazati klijent, koju dobija od servera. Ova poruka se prikazuje kada lokalno menjate podatke, a ne kada ih pošaljete nazad serveru.



SLIKA 21.2 *Poruka o grešci koju prikazuje primer ThinCli2 kada je vrednost polja Area premala*

Status slogova

Komponenta ClientDataSet sadrži karakteristiku koja nam omogućava da nadgledamo šta se dešava u klijent/server paketima podataka; to je metod `UpdateStatus`, koji kao rezultat daje jedan od sledećih indikatora za aktuelni slog:

```
type
  TUpdateStatus = (usUnmodified, usModified,
    usInserted, usDeleted);
```

Da bismo lako proverili status svakog sloga klijent skupa podataka, možemo dodati string-izračunato polje tabeli i izračunati njegovu vrednost sledećim metodom:

```
procedure TForm1.ClientDataSet1CalcFields(
  DataSet: TDataSet);
begin
  ClientDataSet1Status.AsString :=
    GetEnumName (TypeInfo(TUpdateStatus),
      Integer (ClientDataSet1.UpdateStatus));
end;
```

Ovaj metod pretvara trenutnu vrednost enumeracije `TUpdateStatus` u string.

Slogovi se premeštaju od servera do klijenta u zavisnosti od vrednosti svojstva `PacketRecords` komponente ClientDataSet, koja određuje broj slogova po jednom paketu. Unapred određena vrednost svojstva je 5, što znači da će provajder smestiti pet slogova u svaki paket koji šalje. Alternativno, možete odrediti vrednost nula da biste od servera zatražili samo opise polja bez podataka. Na drugom kraju spektra, kada koristite -1 poslaćete sve podatke odjednom (što ima smisla samo za mali skup podataka).

Pristupanje delti

Pored proučavanja statusa svakog sloga, najbolji način da razumete koje promene su se dogodile u datoj ClientDataSet komponenti (ali nisu prosleđene serveru) jeste da pogledate deltu, spisak promena koje čekaju da se primene na serveru. Ovo svojstvo je definisano na sledeći način:

```
property Delta: OleVariant;
```

Format koji se koristi za `Delta` svojstvo je isti format koji se koristi za prosleđivanje podataka od klijenta do servera. Ono što možemo da učinimo je da dodamo još jednu ClientDataSet komponentu aplikaciji i povežemo je sa podacima `Delta` svojstva prvog skupa podataka klijenta:

```
procedure TForm1.ButtonDeltaClick(Sender: TObject);
begin
  if ClientDataSet1.ChangeCount > 0 then
  begin
    ClientDataSet2.Data := ClientDataSet1.Delta;
    ClientDataSet2.Open;
    FormDelta.DataSource1.DataSet := ClientDataSet2;
    FormDelta.Show;
  end
  else
    FormDelta.Hide;
end;
```

FormDelta je veoma jednostavan formular koji sadrži DataSource i DBGrid komponente, kao što možete videti na slici 21.3. Primetićete da delta skup podataka sadrži dva elementa za svaki izmenjeni slog: originalne vrednosti i izmenjena polja.

SLIKA 21.3 *Primer ThinCli2 Vam omogućava da pogledate zahteve za privremenim ažuriranjem koji se čuvaju u Delta svojstvu komponente ClientDataSet*

Ovaj kod prikazuje slogove koji se nalaze u delti, ali ih neće automatski osvežiti kada korisnik načini druge izmene, izuzev kada program kopira novu deltu u drugu ClientDataSet komponentu svaki put kada se podaci izmene.

Ovo možete obaviti u obradi `AfterPost` događaja ClientDataSet komponente koja se izvršava kada se podaci izmene u memoriji, ali ne i kada se pošalju serveru:

```
procedure TForm1.ClientDataSet1AfterPost(DataSet: TDataSet);
begin
  if FormDelta.Visible and
    (ClientDataSet1.ChangeCount > 0) then
  begin
    ClientDataSet2.Data := ClientDataSet1.Delta;
  end;
end;
```

Ažuriranje podataka

Sada kada bolje razumemo šta se dešava prilikom lokalnih ažuriranja, možemo pokušati da učinimo da ovaj program funkcioniše slanjem lokalnog ažuriranja (koje se čuva u delti) nazad do aplikacije servera. Da biste odjednom primenili sva ažuriranja skupa podataka, prosledite `-1` metodu `ApplyUpdates`:

```
procedure TForm1.ButtonUpdateClick(Sender: TObject);
begin
  ClientDataSet1.ApplyUpdates (-1);
  FormDelta.Hide;
end;
```

Operacija ažuriranja može da inicira događaj `OnReconcileError` koji nam omogućava da izmenimo parametar `Action` (koji se prosleđuje po referenci); ova vrednost odmah određuje kako se server ponaša u slučaju kolizije ažuriranja:

```
procedure TForm1.ClientDataSet1ReconcileError(
  DataSet: TClientDataSet; E: EReconcileError;
  UpdateKind: TUpdateKind; var Action: TReconcileAction);
```

Ovaj metod sadrži tri parametra: klijent komponentu skupa podataka (u slučaju da više od jedne klijent aplikacije komunicira sa aplikacijom servera), izuzetak koji je prouzrokovao grešku (sa porukom o grešci) i vrstu operacije koja nije uspela (`ukModify`, `ukInsert` ili `ukDelete`). Povratna vrednost, koju čuvate u parametru `Action`, može biti nešto od sledećeg:

```
type
  TReconcileAction = (raSkip, raAbort, raMerge,
    raCorrect, raCancel, raRefresh);
```

- Vrednost `raSkip` označava da server treba da preskoči slog koji izaziva konflikt, ostavljajući podatke u delti (a to je unapred određena vrednost).
- Vrednost `raAbort` govori serveru da prekine celu operaciju ažuriranja i da čak ne primeni preostale izmene koje se nalaze u delti.
- Vrednost `raMerge` govori serveru da spoji podatke klijenta sa podacima na serveru, primenjujući samo izmenjena polja klijenta (i da zadrži polja koja su izmenili drugi korisnici).
- Vrednost `raCorrect` govori serveru da zameni svoje podatke podacima sa klijenta, zanemarujući sve izmene koje su već obavili drugi klijenti.

- Vrednost `raCancel` ćete koristiti za odustajanje od zahteva za ažuriranjm, uklanjajući element iz delte i restaurirajući vrednosti koje su originalno dobijene iz baze podataka (a time ćete ignorisati izmene koje su načinili drugi korisnici).
- Vrednost `raRefresh` govori serveru da odbaci sva ažuriranja delte klijenta i da ih zameni vrednostima koje se trenutno nalaze na serveru (a time se čuvaju izmene koje su načinili drugi korisnici).

Ukoliko želite da testirate kolizije na jednom računaru, možete jednostavno pokrenuti dve kopije klijent aplikacije, izmeniti isti slog u oba klijenta, a zatim iz oba klijenta poslati izmene. Ovo ćemo kasnije uraditi da bismo generisali grešku, ali hajde da prvo vidimo kako da obradimo događaj `OnReconcileError`.

Ovo je, zapravo, jednostavno postići, ali samo zato što imamo pomoć. Pošto je izrada specifičnog formulara za obradu događaja `OnReconcileError` veoma uobičajena, Delphi već obezbeđuje takav formular u Object Repositoryju. Jednostavno otvorite stranu Dialogs i odaberite element Reconcile Error Dialog. Kako pokazuje izvorni kod ove jedinice, izvozi se funkcija koju direktno možete upotrebiti za inicijalizaciju i prikazivanje okvira za dijalog:

```
procedure TForm1.ClientDataSet1ReconcileError(DataSet: TClientDataSet;
  E: EReconcileError; UpdateKind: TUpdateKind;
  var Action: TReconcileAction);
begin
  Action := HandleReconcileError (DataSet, UpdateKind, E);
end;
```

UPOZORENJE

Kako nagoveštava izvorni kod jedinice Reconcile Error Dialog, potrebno je da upotrebite okvir za dijalog Project Options da biste uklonili ovaj formular iz liste formulara koji se automatski kreiraju (ukoliko to ne učinite, desiće se greška prilikom kompajliranja projekta). Naravno, ovo je potrebno da uradite samo ukoliko niste podesili Delphi tako da preskoči automatsko kreiranje formulara. ■

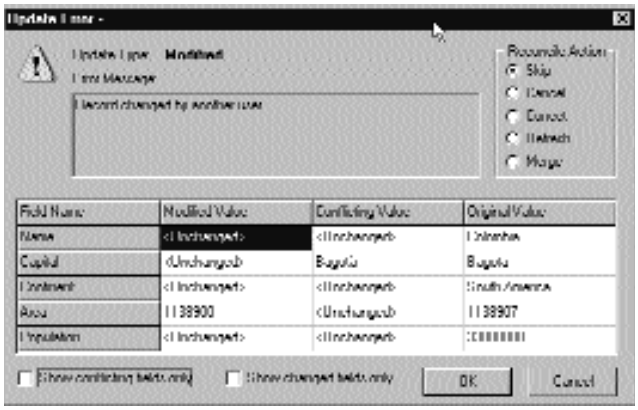
Funkcija `HandleReconcileError` jednostavno kreira formular okvira za dijalog i prikazuje ga:

```
function HandleReconcileError(DataSet: TDataSet;
  UpdateKind: TUpdateKind; ReconcileError: EReconcileError):
  TReconcileAction;
var
  UpdateForm: TReconcileErrorForm;
begin
  UpdateForm := TReconcileErrorForm.CreateForm(DataSet,
    UpdateKind, ReconcileError);
  with UpdateForm do
  try
    if ShowModal = mrOK then
    begin
      Result := TReconcileAction(ActionGroup.Items.Objects[
        ActionGroup.ItemIndex]);
      if Result = raCorrect then
        SetFieldValues(DataSet);
    end
    else
```

```
      Result := raAbort;
    finally
      Free;
    end;
  end;
```

Jedinica Reconc, koja sadrži okvir za dijalog Reconcile Error, sadrži više od 350 linija koda, te je ne možemo detaljno opisati. Ipak, trebalo bi da možete da razumete izvorni kod ukoliko ga pažljivo proučite. Alternativno, možete je koristiti a da Vas ne interesuje kako sve funkcioniše.

Okvir za dijalog će se prikazati u slučaju greške, prikazujući zahtevanu izmenu koja je dovela do konflikta i omogućavajući korisniku da odabere jednu od mogućih `TReconcileAction` vrednosti. Primer možete videti na slici 21.4.



SLIKA 21.4 *Okvir Reconcile Error koji obezbeđuje Delphi, a nalazi se u Object Repositoryju i koristi se u primeru ThinCli2*

Sekvenca ažuriranja

Kao rezime sledi sekvenca operacija koje se odnose na ažuriranje i moguće događaje greške:

1. Klijent program poziva metod `ApplyUpdates` komponente ClientDataSet.
2. Delta se šalje provajderu koji se nalazi u središnjem čvoru. Provajder inicira događaj `OnUpdateData`, u kome možete pogledati zahtevane izmene pre nego što stignu do servera baze podataka. U ovom trenutku možete izmeniti deltu, koja se prosleđuje u formatu koji je kompatibilan sa podacima komponente ClientDataSet.
3. Provajder (deo provajdera koji se naziva resolver — određivač) primenjuje svaki red delte na server baze podataka. Pre primene svakog ažuriranja, provajder prihvata događaj `BeforeUpdateRecord`. Ukoliko ste podesili zastavicu `ResolveToDataSet`, ovo ažuriranje će neizbežno inicirati lokalne događaje skupa podataka središnjeg čvora.

4. U slučaju greške na serveru, provajder inicira događaj `OnUpdateError` (na središnjem čvoru) i program ima šansu da na tom nivou popravi grešku.
5. Ukoliko program središnjeg čvora ne popravi grešku, odgovarajući zahtev za ažuriranjem ostaje u delti. Greška se u tom trenutku prijavljuje klijent strani ili kada se desi određeni broj grešaka, već prema vrednosti parametra `MaxErrors` poziva `ApplyUpdates`.
6. Na kraju, delta paket sa preostalim ažuriranjima se vraća nazad klijentu, čime se inicira događaj `OnReconcileError` komponente ClientDataSet za svako od njih. U ovoj obradi događaja klijent program može pokušati da popravi problem (verovatno tražeći pomoć od korisnika), menjajući ažuriranje u delti i kasnije ga ponovo koristeći.

Osvežavanje podataka

Možete dobiti ažuriranu verziju podataka, koju su drugi korisnici možda izmenili, pozivanjem metoda `Refresh` komponente ClientDataSet. Ipak, ova operacija se može obaviti samo ukoliko korisnik u kešu nema operacije ažuriranja koje čekaju da se obave, jer pozivanje metoda `Refresh` uklanja sve što se nalazi u `Delta` svojstvu. Ukoliko postoje ažuriranja koja čekaju izvršavanje, umesto metoda `Refresh` možete pozvati `RefreshRecords`, koji pokušava da ponovo primeni izmene koje imate u dnevniku delte na nove podatke koji su na raspolaganju u središnjem čvoru, pozivajući `OnReconcileError` u slučaju greške (to jest, ukoliko je jedan od slogova koji se nalaze na klijentu već izmenjen na serveru baze podatka).

Dodavanje opcije Undo

Pošto se ažurirani podaci čuvaju u lokalnoj memoriji (delta), pored primenjivanja ažuriranja i njihovog slanja server aplikaciji, možemo ih odbaciti, uklanjajući primer tom elementu delte. Komponenta ClientDataSet sadrži specifičan metod `UndoLastChange` kojim se ovo postiže. Parametar ovog metoda Vam omogućava da *pratite* operaciju uklanjanja (Undo). Naziv parametra je `FollowChange`. To znači da će se klijent skup podataka pomeriti na slog koji je restauriran operacijom Undo.

Sledi kod koji je povezan sa kontrolom Undo primera:

```
procedure TForm1.ButtonUndoClick(Sender: TObject);
begin
  ClientDataSet1.UndoLastChange (True);
end;
```

Predlažem Vam da sami pokušate da upotrebite ovu karakteristiku da biste u potpunosti shvatili kako funkcioniše.

Podržavanje briefcase modela

Poslednja karakteristika primera ThinCli2 je podrška "briefcase" modelu. Ideja je da ćete klijent program koristiti čak i kada niste fizički povezani sa aplikacijom servera. U tom slučaju, sve podatke koje ćete koristiti možete sačuvati u lokalnom fajlu kada putujete i sa sobom nosite laptop kompjuter (recimo kada posećujete klijente). Klijent program ćete koristiti za pristupanje lokalnoj verziji podataka, normalno ćete ih menjati, a kada se ponovo povežete, primenićete sve izmene koje ste načinili od kada ste prethodni put bili povezani.

Glavni formular primera ThinCli2 sadrži dve kontrole: jednu kojom se čuva zamrznuta slika podataka u lokalnom fajlu i jednu kojom se restauriraju podaci. Obrade `OnClick` događaja ovih kontrola koriste stnadardne komponente OpenDialog i SaveDialog za povezivanje sa fajlom baze podataka:

```
procedure TForm1.ButtonSnapClick(Sender: TObject);
begin
  if SaveDialog1.Execute then
    ClientDataSet1.SaveToFile (SaveDialog1.FileName);
end;
procedure TForm1.ButtonReloadClick(Sender: TObject);
begin
  if OpenDialog1.Execute then
    ClientDataSet1.LoadFromFile (OpenDialog1.FileName);
end;
```

Kada ste završili rad i izmenili lokalni fajl, možete ga ponovo učitati i primeniti izmene na serveru.

Kada koristite briefcase model, najbolje je preuzeti ceo skup podataka pre nego što ga lokalno sačuvate; možda ste zapamtili, to možete učiniti određivanjem vrednosti `-1` za svojstvo Records komponente ClientDataSet. Ukoliko ovo ne učinite, samo ćete sačuvati slogove koji su se trenutno našli u memoriji, a klijent aplikacija neće znati za ostale slogove koji se još uvek nalaze na serveru.

Napredne MIDAS karakteristike

Postoji mnogo više MIDAS karakteristika nego što smo do sada opisali. Ono što sledi je kratak pregled naprednijih karakteristika ove arhitekture, koje delimično pokazuju primeri AppSPlus i ThinPlus. Na nesreću, demonstriranje svake ideje bi ovo poglavlje pretvorilo u celu jednu knjigu (a ne može svaki programer sebi da priušti MIDAS), te ću se ograničiti samo na pregled karakteristika.

Pored karakteristika koje će se razmatrati u narednim odeljcima, primeri AppSPlus i ThinPlus pokazuju upotrebu povezivanja priključaka, ograničenog zapisivanja u dnevnik događaja i ažuriranja na strani servera, i direktno dobijanje sloga na strani klijenta. Poslednja karakteristika se postiže sledećim pozivom:

```
procedure TClientForm.ButtonFetchClick(Sender: TObject);
begin
  ButtonFetch.Caption := IntToStr (cds.GetNextPacket);
end;
```

Ovim Vam je omogućeno da dobijete više slogova nego što je potrebno korisničkom interfejsu klijenta (DbGrid). Drugim rečima, možete direktno pristupati metodima, a da ne morate da čekate akciju korisnika. Preporučujem Vam da proučite ove složene primere pošto pročitate ostatak ovog odeljka.

Parametarski upiti

Ukoliko u upitima ili uskladištenim procedurama želite da koristite parametre, umesto izrade standardnog rešenja (kada metod poziva server), možete dozvoliti Delphiju da Vam pomogne. Prvo u središnjem čvoru definišite upit koji sadrži parametar, npr. ovako:

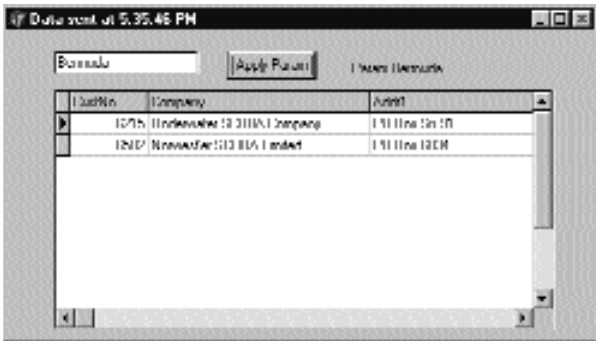
```
select * from customer
  where Country = :Country
```

Upotrebite svojstvo `Params` da biste odredili tip i unapred određenu vrednost parametra. Na strani klijenta možete upotrebiti komandu Fetch Params kontekst menija komponente ClientDataSet, pošto ste je povezali sa odgovarajućim provajderom. U vreme izvršavanja možete pozvati ekvivalent, a to je metod `FetchParams` komponente ClientDataSet.

Sada možete da obezbedite lokalnu unapred određenu vrednost za parametar svojstvom `Params`. Ona će biti poslata središnjem čvoru kada dobijete podatke. Primer ThinPlus osvežava parametar sledećim kodom:

```
procedure TFormQuery.btnParamClick(Sender: TObject);
begin
  cdsQuery.Close;
  cdsQuery.Params[0].AsString := EditParam.Text;
  cdsQuery.Open;
end;
```

Sekundarni formular ovog primera, koji prikazuje rezultate parametarskog upita, možete videti na slici 21.5. Na slici takođe možete videti podatke koje je poslao server, kao što je objašnjeno u odeljku "Prilagođavanje paketa podataka".



SLIKA 21.5 *Sekundarni formular primera ThinPlus prikazuje podatke parametarskog upita*

Praktični pozivi metoda

Kako server sadrži normalni COM interfejs, možemo mu dodati još svojstava i metoda i pozivati ih sa klijenta. Jednostavno otvorite editor biblioteke tipa servera i upotrebite ga kao i bilo koji drugi COM server. U primeru AppSPlus ja sam dodao metod `Login` narednom implementacijom:

```
procedure TAppServerPlus.Login(
  const Name, Password: WideString);
begin
  // TODO: add actual login code...
  if Password <> Name then
    raise Exception.Create (
      'Wrong name/password combination received')
  else
    Query.Active := True;
  ServerForm.Add ('Login:' + Name + '/' + Password);
end;
```

Program izvršava jednostavan test, umesto da proverava kombinaciju ime/lozinka prema listi autorizacija kako bi to trebalo da učini prava aplikacija. Takođe, onemogućavanje Queryja ne funkcioniše, jer ga može aktivirati provajder. Onemogućavanje DataSetProvidera je daleko robusnije rešenje. Klijent može da pristupi serveru na jednostavan način, svojstvom `AppServer` udaljene komponente za povezivanje. Sledi primer poziva primera ThinPlus koji se obavlja u događaju `AfterConnect` komponente za povezivanje:

```
procedure TClientForm.ConnectionAfterConnect(
  Sender: TObject);
begin
  Connection.AppServer.Login (Edit2.Text, Edit3.Text);
end;
```

Imajte na umu da možete pozivati dodatne metode COM interfejsa preko DCOM-a i CORBA, a takođe i upotrebom HTTP veze. Kako program koristi `safecall` konvenciju pozivanja, izuzetak koji se javlja na serveru se automatski prosleđuje i prikazuje na strani klijenta. Na ovaj način, kada korisnik selektuje polje za potvrdu Connect, obrada događaja koja se koristi za uključivanje klijent skupova podataka se prekida, a korisnik sa pogrešnom lozinkom neće moći da pogleda podatke.

Pored direktnih poziva metoda od klijenta ka serveru, možete implementirati i povratne pozive od servera ka klijentu. Ovo se, na primer, može upotrebiti za obaveštavanje svakog klijenta o specifičnim događajima. Upotreba COM događaja je način da ovo postignete. Alternativno, možete dodati novi interfejs, koji implementira klijent, koji serveru prosleđuje objekat implementacije. Na ovaj način server može da pozove metod koji je na klijent kompjuteru. Povratni pozivi nisu mogući kada je u pitanju HTTP veza. Sa povezivanjem na bazi priključaka, povratni pozivi su mogući samo kada je instaliran WinSock2, što uključuje kompjutere sa operativnim sistemom Windows 98 i kompjutere sa operativnim sistemom Windows 95 koji imaju instaliranu poslednju verziju Internet Explorera.

Master/detail zavisnosti

Ukoliko Vaša aplikacija u središnjem čvoru izvozi vše skupova podataka, možete ih dobiti upotrebom više ClientDataSet komponenata na strani klijenta i možete ih lokalno povezati sa master/detail strukturom. Ovo će stvoriti prilično problema lokalnim skupovima podataka izuzev ukoliko ne prihvatite sve slogove.

Ovo rešenje čini ažuriranje prilično komplikovanim; obično ne možete da poništite master slog dok se ne uklone svi odgovarajući detail slogovi, i ne možete da dodate detail slogove sve dok se master slog ne nađe na pravom mestu. (Zapravo, različiti serveri ovo rešavaju na različite načine, ali u većini slučajeva kada se koristi strani ključ, ovo je standardno ponašanje.) Da biste rešili ovaj problem, možete da napišete komplikovan kod na strani klijenta za ažuriranje slogova dveju tabela prema specifičnim pravilima.

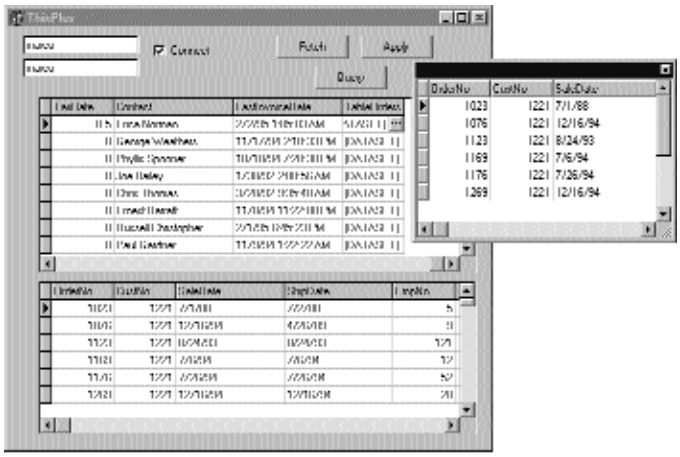
Potpuno drugačiji pristup je da dobijete jedan skup podataka koji već sadrži detalje u polju skupa podataka, polju tipa `TDatasetField`. Da biste ovo postigli, potrebno je da podesite master/detail zavisnost na server aplikaciji:

```
object TableCustomer: TTable
  DatabaseName = 'DBDEMOS'
  TableName = 'customer.db'
end
object TableOrders: TTable
  DatabaseName = 'DBDEMOS'
  MasterFields = 'CustNo'
  MasterSource = DataSourceCust
  TableName = 'ORDERS.DB'
end
object DataSourceCust: TDataSource
  DataSet = TableCustomer
end
object ProviderCustomer: TDataSetProvider
  DataSet = TableCustomer
end
```

Na strani klijenta detail tabela će prikazati dodatno polje komponente ClientDataSet, a kontrola DbGrid će ga prikazati kao dodatnu kolonu koja je označena trima tačkama. Kada kliknete kontrolu prikazaće se sekundarni formular u kojem se prikazuje detail tabela (videti sliku 21.6). Ukoliko je potrebno da izradite fleksibilan korisnički interfejs na strani klijenta, tada možete da dodate sekundarnu komponentu ClientDataSet koja je povezana sa poljem skupa podataka master skupa podataka upotrebom svojstva `DataSetField`. Jednostavno kreirajte stalna polja za glavnu komponentu ClientDataSet i zatim povežite svojstvo:

```
object cdsDet: TClientDataSet
  DataSetField = cdsTableOrders
end
```

Ovakvim povezivanjem možete prikazati detail skup podataka u zasebnoj komponenti DbGrid, koja se kao i obično nalazi na formularu (donji deo slike 21.6), ili na bilo koji drugi način koji želite. Primetite da se sa ovakvom strukturom ažuriranja odnose samo na master tabelu i da bi trebalo da server obradi pravilnu sekvencu ažuriranja čak i u složenim situacijama.



SLIKA 21.6 *Primer ThinPlus pokazuje kako se polje podataka može prikazati u tabeli u pokretnom prozoru ili se može izdvojiti komponentom ClientDataSet i prikazati u sekundarnom formularu. Obično ćete učiniti jednu od ove dve stvari, ne obe!*

Još opcija provajdera

Već sam pomenuo svojstvo `Options` komponente `DataSetProvider` — ništa što se može koristiti za dodavanje svojstava polja paketu podataka. Postoji nekoliko drugih opcija koje možete upotrebiti za prilagođavanje paketa podataka i za prilagođavanje ponašanja klijent programa. Evo kratke liste:

- Možete minimizovati preuzimanje BLOB podataka opcijom `poFetchBlobsOnDemand`. U ovom slučaju, klijent aplikacija može preuzimati BLOB-ove određivanjem vrednosti True za svojstvo `FetchOnDemand` komponente `ClientDataSet` ili pozivanjem metoda `Fetchblob` za određene slogove. Slično, možete onemogućiti automatsko preuzimanje detail slogova određivanjem opcije `poFetchDetailOn Demand`. Ponoviću, klijent može upotrebiti svojstvo `FetchOnDemand` ili pozvati metod `Fetchdetails`.

- Kada koristite master/detail zavisnost, možete kontrolisati kaskade jednom od dve opcije. Zastavica `poCascadeDeletes` kontroliše da li provajder treba da obriše detail slogove pre uklanjanja master sloga. Ovu opciju možete podesiti ukoliko server baze podataka obavlja kaskadna uklanjanja za Vas kao deo njegove podrške referencijalnom integritetu. Slično, možete podesiti opciju `poCascadeUpdates` kada ažuriranje ključnih vrednosti master/detail zavisnosti može automatski biti obavljeno na serveru.

- Možete ograničiti operacije na klijent strani. Najrestriktivnija opcija je `poReadOnly`, koja onemogućava bilo kakvo ažuriranje. Ukoliko korisniku želite da dodelite ograničenu mogućnost izmena, takođe možete upotrebiti `poDisableInserts`, `poDisableEdits` ili `poDisableDeletes`.

- Možete ponovo poslati klijet kopiju slogova koje je klijent izmenio upotrebom opcije `poAutoRefresh`, što je korisno u slučaju da su drugi korisnici simultano načnili izmene koje nisu dovele do konflikta. Takođe, možete poslati nazad klijentu izmene koje su načinjene obradama događaja `BeforeUpdaterecord` ili `AfterUpdaterecord` određivanjem opcije `poPropagateChanges`. Ova opcija je, takođe, zgodna kada koristite polja sa automatskim uvećavanjem njihove vrednosti, okidače (triggers) i druge tehnike kojima se menjaju podaci na serveru ili središnjem čvoru posle izmena koje zahteva klijent.

Na kraju, ukoliko želite da klijent upravlja ovim operacijama, možete aktivirati opciju `poAllowCommandText`. Na ovaj način možete odrediti SQL upit ili naziv tabele središnjeg čvora iz klijenta upotrebom metoda `GetRecords` ili metoda `Execute`.

Simple Object Broker

Komponenta SimpleObjectBroker obezbeđuje lak način za pronalaženje aplikacije servera između nekoliko server kompjutera. Vi samo obezbeđujete spisak kompjutera koji su na raspolaganju, a klijent će probati svaki od njih sve dok ne nađe onaj koji je na raspolaganju.

Dalje, ukoliko aktivirate svojstvo `LoadBalanced`, komponenta će nasumično birati jedan od servera; kada mnogo klijenata koristi istu konfiguraciju, veza će biti automatski distribuirana između više servera. Ukoliko Vam ovo izgleda loše, razmislite o tome da neki veoma skupi sistemi za balansiranje učitavanja zapravo i ne nude mnogo više od ovoga.

Prozivanje objekata

Kada se više klijenata poveže istovremeno sa Vašim serverom, na raspolaganju imate dve opcije. Prva je da kreirate udaljeni objekat modula podataka za svaki od njih i da omogućite da svaki zahtev bude obrađen u nizu (unapred određeno ponašanje COM servera kada se koristi stil `ciMultiInstance`). Alternativno, možete prepustiti sistemu da kreira različite instance aplikacije za svakog od klijenata (`ciSingleInstance`). Ovaj način zahteva više resursa i više SQL server veza (i licenci), sa mogućnošću preopterećenja BDE-a (u Poglavlju 17 smo videli da BDE ne može obraditi više od skupa procesa).

Alternativni pristup je ponuđen podrškom u tehnologiji MIDAS 3 za prozivanje (pooling) objekata. Sve što je potrebno da zahtevate ovu karakteristiku je da dodate poziv `RegisterPooled` u metodu `UpdateRegistry`. U kombinaciji sa podrškom bez stanja koja je sada ugrađena u MIDAS, mogućnost prozivanja Vam omogućava da delite neke objekte središnjeg čvora između mnogih klijenata.

Korisnici klijent kompjutera će veći deo vremena provesti čitajući podatke i unoseći nove vrednosti i neće nastaviti da traže podatke i šalju ažuriranja. Kada klijent ne poziva metod objekta središnjeg čvora, ovo se može upotrebiti za nekog drugog klijenta. Kada je bez stanja, svaki zahtev dolazi do središnjeg čvora kao potpuno nova operacija, čak i kada je server namenjen određenom klijentu.

Ovaj pooling mehanizam je ugrađen u MTS i CORBA tehnologije, ali MIDAS 3 ga čini dostupnim i za HTTP i veze na osnovu priključaka, kao i za Internet Express Web klijenta.

Prilagođavanje paketa podataka

Postoje mnogi načini da uključite informacije uz paket podataka koji obrađuje `IAppServer` interfejs. Najjednostavniji način je verovatno obrada `OnGetDataSetProperties` događaja samog provajdera. Ovaj događaj sadrži parametar `Sender`, parametar skupa podataka koji naznačava odakle dolaze podaci i niz `OleVariant` parametra `Properties`, u koji možete smestiti dodatne informacije. Potrebno je da definišete niz Variant za svako dodatno svojstvo i da uključite naziv dodatnog svojstva, njegovu vrednost i to da li želite da se podaci vrate na server uz ažuriranu deltu (parametar `IncludeInDelta`).

Naravno, možete prosleđivati svojstva povezane komponente skupa podataka, ali takođe možete proslediti i bilo koju drugu vrednost (dodatna lažna svojstva). U primeru AppSPlus ja sam prosledio klijentu vreme kada je izvršen upit i njegove parametre:

```
procedure TAppServerPlus.ProviderQueryGetDataSetProperties(
  Sender: TObject; DataSet: TDataSet; out Properties: OleVariant);
begin
  Properties := VarArrayCreate([0,1], varVariant);
  Properties[0] := VarArrayOf(['Time', Now, True]);
  Properties[1] := VarArrayOf(['Param',
    Query.Params[0].AsString, False]);
end;
```

Na strani klijenta, komponenta ClientDataSet sadrži metod `GetOptionalParameter` za dobijanje vrednosti dodatnog svojstva zadatog naziva. Komponenta ClientDataSet takođe sadrži metod `SetOptionalParameter` za dodavanje još svojstava skupu podataka. Ove vrednosti će biti sačuvane na disku (u briefcase modelu) i biće poslate nazad središnjem čvoru (određivanjem vrednosti `True` članu `IncludeInDelta` niza). Sledi jednostavan primer dobijanja skupa podataka u prethodnom kodu:

```
Caption := 'Data sent at ' + TimeToStr (
  TDateTime (cdsQuery.GetOptionalParam('Time')));
Label1.Caption := 'Param ' +
  cdsQuery.GetOptionalParam('Param');
```

Efekat ovog koda je vidljiv na slici 21.5. Alternativni i moćniji pristup za prilagođavanje paketa podataka koji se šalju klijentu je obrada događaja `OnGetdata` provajdera, koji prihvata paket podataka u formi skupa podataka klijenta. Upotrebom metoda ovog skupa podataka klijenta, podatke možete izmeniti pre nego što se pošalju klijentu. Na primer, možete enkodirati neke podatke ili izbaciti osetljive slogove.

Sakrivena snaga komponente ClientDataSet

Komponenta ClientDataSet podržava mnoge karakteristike, od kojih se neke odnose na višelinijsku arhitekturu, ali se takođe mogu upotrebiti i u nekim drugim prilikama. Ove komponente predstavljaju bazu podataka koja je kompletno mapirana u memoriji, a ovo čini mogućim obavljanje operacija u hodu, kao što je kreiranje indeksa, koje drugi skupovi podataka obično ne podržavaju. Da biste sortirali upit, Vi ga obično ponovo izvršavate. Da biste indeksirali lokalnu tabelu, potrebno je da je indeks definisan. Samo SDO skupovi podataka imaju nešto mogućnosti dinamičkog indeksiranja kakvo postoji za komponentu ClientDataSet.

Indeksiranje nije sve što komponenta ClientDataSet ima da ponudi. Kada imate indeks, možete definisati grupe na osnovu indeksa, sa mogućnošću postojanja više nivoa grupisanja. Postoji čak i specifična podrška za određivanje pozicije sloga u okviru grupe (prvi, poslednji ili negde u sredini). Nad grupama ili celim tabelama možete definisati sume, to jest, možete izračunati ukupnu ili prosečnu vrednost kolone za celu tabelu ili za aktuelnu grupu. Podaci ne moraju da se pošalju serveru jer se ove agregatne operacije obavljaju u memoriji. Možete čak definisati nova agregatna polja, za koja možete direktno povezati kontrole koje prepoznaju podatke.

Važna stvar koju treba imati na umu je da su sve ove karakteristike na raspolaganju ne samo za MIDAS aplikacije već i za klijent/server aplikacije, pa čak i za lokalne aplikacije. Komponenta ClientDataSet svoje podatke može dobiti sa udaljene MIDAS veze, od lokalnog skupa podataka (kreirajući zamrznutu sliku podataka), ili iz lokalnog fajla (kao kod briefcase modela, ali kada je cela tabela definisana samo u okviru klijent skupa podataka).

Ovo je još jedna ogromna oblast koju treba istražiti, te ću Vam samo pokazati nekoliko primera koji ističu ključne karakteristike. Ovi primeri neće biti zasnovani na MIDAS-u već na lokalnim tabelama.

Definisanje apstraktnih tipova podataka

Interesantna karakteristika VCL podrške baze podataka koju možete aktivirati kada koristite komponentu ClientDataSet nad lokalnim fajlom, jeste definisanje apstraktnih tipova podataka. Jednostavno, smestite komponentu ClientDataSet na formular, aktivirajte editor za svojstvo `FieldDefs`, dodajte nekoliko polja i odaberite vrednost `ftADT` za svojstvo `DataType` jednog od polja. Sada pređite na svojstvo `ChildDefs` i definišite dete polja. Sledi definicija primera AdtDemo:

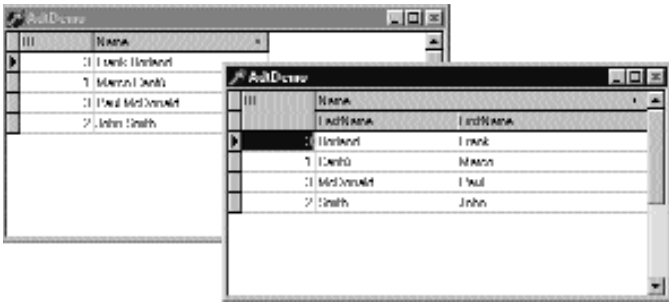
```
FieldDefs = <
  item
    Name = 'ID'
    DataType = ftInteger
  end
  item
    Name = 'Name'
    ChildDefs = <
      item
        Name = 'LastName'
        DataType = ftString
        Size = 20
      end
      item
        Name = 'FirstName'
        DataType = ftString
        Size = 20
      end>
    DataType = ftADT
    Size = 2
  end>
```

U ovom trenutku jednostavno unesite naziv za svojstvo `FileName` komponente ClientDataSet, kliknite desnim tasterom miša komponentu i odaberite komandu Create Table; spremni ste da kompajlirate i pokrenete aplikaciju (posle povezivanja komponenata koje prepoznaju podatke).

Podaci će automatski biti izdvojeni iz fajla koji ste obezbedili, a izmene će biti sačuvane u fajlu kada zatvorite program.

Ukoliko koristite komponentu DBGrid za prikazivanje rezultujućeg skupa podataka, omogućiće Vam da prikažete ili sakrijete potpolja ADT polja, kao što možete videti na slici 21.7. Možete obezbediti kondenzovanu vrednost polja definisanjem njegovog događaja `OnGetText` (u Delphiju 4 je postojala unapred određena vrednost, ali ona nije na raspolaganju u Delphiju 5):

```
procedure TForm1.ClientDataSet1NameGetText(Sender: TField;
  var Text: String; DisplayText: Boolean);
begin
  Text := ClientDataSet1NameFirstName.AsString + ' ' +
    ClientDataSet1NameLastName.AsString;
end;
```



SLIKA 21.7 *Primer AdtDemo pokazuje podršku za prikazivanje i sakrivanje definicije ADT polja*

Indeksiranje "u hodu"

Kada imate podatke u komponenti ClientDataSet, podaci se u potpunosti nalaze u memoriji. Kada komponentu zasnivamo na lokalnom fajlu, kao u primeru AdtDemo, ceo fajl se učitava u memoriju kada se pokrene program. Ovo se razlikuje od, recimo, Paradox tabele, za koju BDE učitava samo polja kojima se pristupa.

Prednost koju imate kada je cela tabela u memoriji, jeste u tome da je možete prilično brzo sortirati. Upotrebom komponente ClientDataSet to možete učiniti jednostavnim dodeljivanjem odgovarajućeg naziva svojstvu `IndexFieldNames`. U primeru AdtDemo (kao i u mnogim programima) ova izmena indeksa se obavlja kada kliknete zaglavlje kontrole DBGrid (čime se inicira događaj `OnTitleClick`):

```
procedure TForm1.DBGrid1TitleClick(Column: TColumn);
begin
  if Column.Field.FullName = 'Name' then
    ClientDataSet1.IndexFieldNames := 'Name.LastName'
  else
    ClientDataSet1.IndexFieldNames := Column.Field.FullName;
end;
```

Program koristi svojstvo `FullName` polja (ne svojstvo `FieldName`) zbog ADT definicije. Za dete polja, zapravo, indeks bi trebalo da se izradi na osnovu `Name.LastName`, a ne jednostavno nad `LastName`. Takođe, ADT polje ne može biti indeksirano, te ukoliko je selektovano, program kao indeks koristi potpolje `LastName`. Ovi indeksi nisu stalni; ne čuvaju se u fajlu već se jednostavno primenjuju u memoriji.

SAVET

Komponenta ClientDataSet može imati indeks na osnovu polja koje se izračunava, preciznije nad internim poljem koje se izračunava, tipom polja koje je na raspolaganju samo za ovaj skup podataka. ■

Grupisanje

Kada ste definisali indeks za komponentu ClientDataSet, možete grupisati podatke prema tom indeksu. Praktično, grupa je definisana kao lista uzastopnih polja (prema indeksu) za koja se vrednost indeksiranog polja ne menja. Na primer, ukoliko imate indeks nad državom, sve adrese koje su u toj državi će pripadati istoj grupi.

Primer CdsCacls sadrži komponentu ClientDataSet koja izdvaja svoje podatke iz tabele Country poznate baze podataka DBDEMOS. Ova operacija se može izvršiti u vreme dizajniranja upotrebom komande Assign Local Data iz kontekst menija komponente ClientDataSet. Da biste izdvojili podatke u vreme izvršavanja i dobili ažuriranu sliku, možete dodati formularu komponentu DataSetProvider povezujući tri komponente na sledeći način:

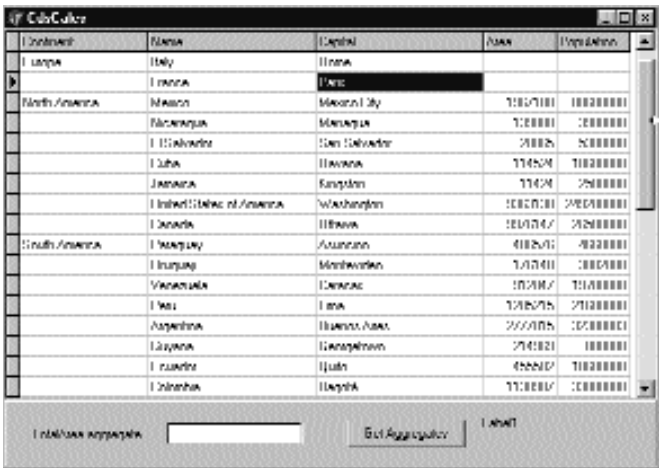
```
object Table1: TTable
  Active = True
  DatabaseName = 'DBDEMOS'
  TableName = 'COUNTRY.DB'
end
object DataSetProvider1: TDataSetProvider
  DataSet = Table1
end
object ClientDataSet1: TClientDataSet
  ProviderName = 'DataSetProvider1'
end
```

Sada se možemo posvetiti definiciji grupe. Ovo se ostvaruje, pored definisanja indeksa, navođenjem nivoa grupisanja za sam indeks:

```
object ClientDataSet1: TClientDataSet
  IndexDefs = <
    item
      Name = 'ClientDataSet1Index1'
      Fields = 'Continent'
      GroupingLevel = 1
    end>
  IndexName = 'ClientDataSet1Index1'
```

Kada imate aktivnu grupu, možete ovo prikazati korisniku dajući mu uvid u strukturu grupisanja komponente DBGrid kao što je prikazano na slici 21.8. Jednostavno obradite događaj `OnGetText` za grupisano polje (`Continent` u našem primeru) i prikažite tekst samo ukoliko je slog prvi u grupi:

```
procedure TForm1.ClientDataSet1ContinentGetText(Sender: TField;
  var Text: String; DisplayText: Boolean);
begin
  if gbFirst in ClientDataSet1.GetGroupState (1) then
    Text := Sender.AsString
  else
    Text := '';
end;
```



SLIKA 21.8 *Primer CdsCalcs pokazuje da pisanjem koda zaglavlja kontrola DBGrid može vizuelno prikazati grupisanje definisano u komponenti ClientDataSet*

Definisanje suma

Još jedna zaista moćna karakteristika komponente ClientDataSet je podrška sumiranju (*aggregates*). Suma je izračunata vrednost zasnovana na više slogova, kao što je ukupna ili prosečna vrednost polja za celu tabelu ili grupu slogova (koja je definisana logikom grupisanja koju sam upravo opisao). Sume se održavaju, to jest, one se ponovo izračunavaju ukoliko se jedan od slogova promeni. Na primer, ukupna vrednost narudžbenice će se automatski održavati dok korisnik unosi elemente narudžbenice.

NAPOMENA

Sume se sukcesivno održavaju, ali ne ponovnim izračunavanjem svih vrednosti svaki put kada se promeni jedna vrednost. Ažuriranje suma koristi prednost delti koje prati komponenta ClientDataSet. Na primer, da bi se ažuriralo Sum kada se promeni vrednost polja, komponenta ClientDataSet oduzima staru vrednost, a zatim dodaje novu vrednost. Potrebna su samo dva izračunavanja, čak i kada postoje hiljade redova u toj agregatnoj grupi. Zbog toga su agregatna ažuriranja momentalna. ■

Postoje dva načina za definisanje suma. Možete upotrebiti svojstvo `Aggregates` komponente ClientDataSet, a to je kolekcija, ili možete definisati agregatna polja upotrebom Fields editora. U oba slučaja definišete agregatne izraze, dodeljujete im naziv i povezujete izraz sa indeksom i

nivoom grupisanja (izuzev ukoliko ne želite da primenite agregatnu funkciju nad celom tabelom). Sledi kolekcija Aggregates primera CdsCalcs:

```
object ClientDataSet1: TClientDataSet
  Aggregates = <
    item
      Active = True
      AggregateName = 'Count'
      Expression = 'COUNT (NAME)'
      GroupingLevel = 1
      IndexName = 'ClientDataSet1Index1'
      Visible = False
    end
    item
      Active = True
      AggregateName = 'TotalPopulation'
      Expression = 'SUM (POPULATION)'
      Visible = False
    end>
  AggregatesActive = True
```

Primetićete da u poslednjoj liniji prethodnog koda morate aktivirati podršku za agregatne funkcije, pored aktiviranja svake agregatne funkcjie koju želite da koristite. Deaktiviranje je takođe važno, jer postojanje previše sumiranja može usporiti program. Alternativni pristup, koji sam ranije pomenuo, jeste upotreba Fields editora i selektovanje komande New Field iz njegovog kontekst menija, a zatim ćete odabrati opciju Aggregate (koja je na raspolaganju uz opciju InternalCalc, samo u komponenti ClientDataSet). Sledi definicija agregatnog polja:

```
object ClientDataSet1: TClientDataSet
  object ClientDataSet1TotalArea: TAggregateField
    FieldName = 'TotalArea'
    ReadOnly = True
    Visible = True
    Active = True
    DisplayFormat = '###,###,###'
    Expression = 'SUM(AREA)'
    GroupingLevel = 1
    IndexName = 'ClientDataSet1Index1'
  end
```

Agregatna polja se prikazuju u Fields editoru u zasebnoj grupi, kao što možete videti na slici 21.9. Prednost upotrebe agregatnog polja, u poređenju sa običnim sumiranjem, jeste u tome da možete definisati format za prikazivanje i da polje možete direktno povezati sa kontrolom koja prepoznaje podatke, kao što je DBEdit u primeru CdsCalcs. Pošto je sumiranje povezano sa grupom, čim selektujete slog neke druge grupe, izlaz će se automatski ažurirati. Takođe, ukoliko promenite podatke, ukupna vrednost će odmah prikazati novu vrednost.

SLIKA 21.9 *Donji deo Fields editora komponente ClientDataSet prikazuje agregatna polja*

Da biste upotrebili obično sumiranje, potrebno je da napišete nešto koda, kao u narednom primeru (primetićete da je vrednost sume tipa Variant):

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption :=
     'Area: ' + ClientDataSet1TotalArea.DisplayText +
    #13'Population : ' + FormatFloat ('###,###,###',
    ClientDataSet1.Aggregates [1].Value) + #13'Number : ' +
    IntToStr (ClientDataSet1.Aggregates [0].Value);
end;
```

Distribuirani servisi visoke klase (MTS i CORBA)

Pored upotrebe običnih priključaka, DCOM-a ili HTTP veze, Delphi i MIDAS podržavaju i dva moćnija transporta višeg nivoa i objekte broker arhitekture. MTS je Microsoftovo rešenje za distribuirano procesiranje, dok je CORBA mnogo češće rasprostranjen kod većine UNIX sistema i često se koristi u Java okruženjima. Za obe teme bi bila potrebna zasebna knjiga ukoliko biste želeli da ih detaljno obradite. Moj cilj ovde je da samo istaknem nekoliko njihovih karakteristika i razmotrim podršku koja postoji u Delphiju.

Microsoftov server transakcija

Pored običnih DCOM servera, Delphi Vam takođe omogućava da kreirate komponente Microsoftovog servera transakcija. Zapravo, možete izraditi obične MTS komponente ili MTS udaljeni modul podataka. U oba slučaja započećete programiranje upotrebom jednog od čarobnjaka koji Vam stoje na raspolaganju u Delphiju. MTS je servis operativnog sistema koji možete instalirati pod Windowsom NT ili Windowsom 98. MTS je, takođe, jedan od temelja Windows 2000 COM+ tehnologije; MTS će verovatno biti utopljen u COM+, a naziv MTS će, eventualno, nestati.

MTS je okruženje u vreme izvršavanja koje obezbeđuje servise transakcija baze podataka, zaštitu, prozivanje resursa i sveukupno poboljšanje robusnosti za DCOM aplikacije. MTS okruženje upravlja objektima koji se nazivaju MTS *komponentama* (MTS components). To su COM objekti koji se čuvaju u serveru u procesu (to jest, u DLL-u). Dok se ostali COM objekti izvršavaju direktno u klijent aplikaciji, MTS objekti se obrađuju u MTS izvršnom okruženju. MTS biblioteke

su instalirane u MTS okruženje. MTS objekti moraju da podrže specifične COM interfejse, počevši od `IObjectControl` interfejsa, koji je osnovni interfejs (kao `IUnknown` za COM objekat).

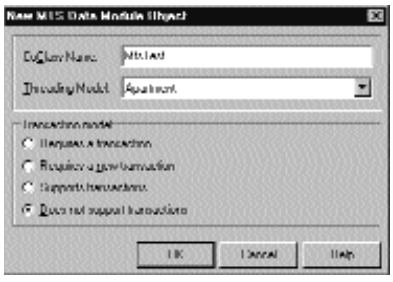
Pre nego što navedemo previše tehničkih detalja niskog nivoa, razmotrimo MTS iz druge perspektive. Kakva je dobit od ovakvog pristupa? MTS obezbeđuje nekoliko interesantnih karakteristika, uključujući i sledeće:

- **SIGURNOST NA OSNOVU ULOGE (ROLE-BASED SECURITY)** Uloga dodeljena klijentu određuje da li ima odgovarajući pristup interfejsu modula podataka.
- **REDUKOVANI RESURSI BAZE PODATAKA** Možete redukovati broj veza sa bazom podataka, jer se središnji čvor prijavljuje na server i koristi iste veze za više klijenata (mada ne možete imati više klijenata odjednom povezanih na server nego što imate licenci). Još više, možete podesiti komponentu tako da će MTS server instancirati module podataka samo za minimum neophodnog vremena.
- **TRANSAKCIJE BAZE PODATAKA** MTS podrška transakcijama uključuje podršku za više baza podataka, mada samo nekoliko SQL servera podržva MTS.

Kreiranje MTS modula podataka

Ukoliko odaberte ikonu MTS Data Module na strani Multitier okvira za dijalog File➥New (u Delphi Enterprise izdanju), lako možete podesiti MTS udaljeni server. MTS Data Module Wizard (videti sliku 21.10) Vam omogućava da unesete naziv za klasu MTS komponente, linijski model (jer MTS serijalizuje sve zahteve, a Single ili Apartment će poslužiti) i model transakcija:

- *Zahteva se transakcija* (Requires a transaction) označava da se svaki poziv od klijenta ka serveru smatra MTS transakcijom (izuzev ukoliko onaj ko vrši poziv ne obezbedi kontekst transakcije).
- *Zathteva se nova transakcija* (Requires a new transaction) označava da se svaki poziv smatra novom MTS transakcijom.
- *Podržava transakcije* (Supports transactions) označava da klijent mora eksplicitno da obezbedi kontekst transakcije.
- *Ne podržava transakciju* (Does not support transaction) — unapred određeno označava da udaljeni modul podataka neće učestvovati u MTS transakciji.



SLIKA 21.10 *Delphijev MTS Data Module Wizard*

Kada ste kreirali MTS modul podataka, lako ga možete izraditi kao što smo to učinili u prethodnim primerima za udaljene module podataka, dodajući komponentu DataSet i izvozeći svojstva njenog provajdera. Takođe, možete dodati svoje metode biblioteci tipa modula podataka. U okviru MTS modula podataka možete upotrebiti metod `GetObjectContext`, koji kao povratnu vrednost daje `IObjectContext` interfejs MTS objekta.

`IObjectContext` interfejs obezbeđuje podršku za transakcije. Možete upotrebiti `SetComplete` da biste MTS okruženju naznačili da je objekat završio rad i da se može deaktivirati, tako da transakcija može da se upiše. Pozovite `EnableCommit` da biste naznačili da objekat nije završio, ali da transakcija treba da se upiše; `DisableCommit` da biste zaustavili operaciju upisivanja čak i kada se metod završio, onemogućavanjem deaktiviranja objekta između dva poziva; `SetAbort` da biste naznačili da je objekat završio i da se može aktivirati, ali da transakcija ne može da se upiše; `IsInTransaction` da biste proverili da li je objekat deo transakcije. Ostali metodi `IObjectContext` interfejsa su `CreateInstance`, kojim se kreira još jedan MTS objekat u istom kontekstu i unutar aktuelne transakcije, `IsCallerInRole`, kojim se proverava da li je onaj koji poziva objekat u određenoj ulozi "zaštite", i `IsSecurityEnabled` (čiji naziv sve govori).

Kada ste izradili MTS biblioteku servera, lako je možete instalirati upotrebom opcije Run➥Install MTS Object. Možete dodati novu biblioteku postojećem MTS paketu (nemojte ovo pomešati sa Delphi paketom komponenata) ili kreirati novi paket iz Delphi okruženja. Kada je MTS objekat instaliran, direktno će biti dostupan ostalim aplikacijama i biće vidljiv u aplikaciji Transaction Server Explorer. Ovaj Microsoftov program za konfigurisanje trebalo bi da je instaliran na Vašem kompjuteru uz MTS podršku.

Kada ste izradili server, možete ga povezati sa klijent aplkacijom koja koristi komponentu DCOMConnection, kao što sam ja učinio u prethodnim primerima. Ovaj kratak uvod bi trebalo da Vam je dao ideju kako da upotrebite MTS za programiranje višelinijskih aplikacija. Prednosti u terminima instalacije, u poređenju sa direktnom upotrebom DCOM-a, su zaista vredne dodatnog truda upotrebe MTS-a.

CORBA

Posle kratkog uvoda u MTS, spremni smo za slično iskustvo kada je u pitanju svet Common Object Request Broker Architecture (CORBA). CORBA standard je definisala Object Managament Group (OMG) i on odgovara složenosti izrade i prosleđivanja distribuiranih aplikacija koje se zasnivaju na objektima.

Jedan od ključnih elemenata CORBA arhitekture je da je ona u potpunosti nezavisna od platformi i operativnih sistema. CORBA Delphi programerima otvara svet koji nema veze sa Microsoftom: mada se moduli izrađeni u Delphiju mogu izvršavati samo pod Windowsom, mogu se povezati sa drugim CORBA objektima koji se izvršavaju na različitim operativnim sistemima. Na primer, CORBA obezbeđuje dobru integraciju sa Javom i možete kupiti Borland/Inprise višelinijsku arhitekturu za Delphi i JBuilder, Borlandovo Java okruženje za programiranje. Zapravo, možete da upotrebite MIDAS CORBA arhitekturu za izradu Delphi klijenata upotrebom Java MIDAS servera pod UNIX-om (ili Linuxom) ili možete izraditi Javu za svoje Windows NT MIDAS servere izrađene uz pomoć Delphija.

CORBA specifikacija definiše kako programi na strani klijenta komuniciraju sa objektima na server strani putem Object Request Brokera (ORB). Inpriseov VisiBroker ORB je broker zahteva koji možete pronaći u svakoj kopiji Delphi Enterprise izdanja. Naravno, potrebna Vam je licenca da biste prosledili aplikacije izrađene upotrebom ovog ORB-a.

Jednostavan CORBA server

Možete izraditi jednostavan CORBA server zasnovan na modulu podataka. Kada odaberete CORBA Data Module opciju na strani Multitier Object Repositoryja, Delphi prikazuje jednostavan CORBA Data Module Wizard koji je prikazan na slici 21.11.

SLIKA 21.11 *CORBA Data Module Wizard Vam omogućava da izradite udaljeni modul podataka uz CORBA podršku*

U ovom čarobnjaku možete odabrati modele instanciranja i model linija. Ovi modeli se razlikuju od modela koje koristi COM:

- Instanciranje *po klijentu* (Per-client) označava da se kreira nova instanca modula podataka za svaku vezu.
- *Deljeno instanciranje* (Shared) označava da jedna instanca modula podataka obrađuje sve zahteve klijenata. Ovaj drugi pristup je moguć samo za protokol bez stanja, što nije teško u MIDAS-u 3.

Kod jednolinijskog servera svaki modul podataka dobija samo jedan zahtev klijenta u jednom trenutku, tako da su podaci instance bezbedni od mogućih konflikata. Kod višelinijskog servera, nasuprot tome, klijent može poslati više simultanih zahteva, a to od programera zahteva dodatnu pažnju.

Kada je kreiran modul podataka, možete nastaviti programiranje kao da je u pitanju bilo koji drugi udaljeni modul podataka. Klasa `TCorbaDataModule` je direktno izvedena iz klase `TRemoteDataModule`. Modul podataka takođe sadrži odgovarajući element u biblioteci tipa, kao što možete videti otvaranjem odgovarajućeg editora ili proučavanjem prevedenog Pascal koda. U prevedenom fajlu Corba1_TLB možete videti `IFirstCorba` interfejs i `IFirstCorba` dispinterfejs, kao i za COM aplikaciju.

Zapravo, u Delphiju 5, CORBA podrška je još uvek ograničena. Glavni ograničavajući faktor Delphijevoj CORBA podršci je to da ona omogućava samo pozive preko CORBA kasnog povezivanja, koje se naziva DII. Ovo je analogno pozivanju COM dispinterfejs metoda preko poziva promenljivih metoda. Zapravo, Delphijevi pozivi CORBA metoda se mogu obaviti preko promenljivih, ali se prosleđuju preko CORBA servisa, ne preko COM-a. Kao i pozivi dispinterfejsa u COM-u, DII u CORBA-i nije najbrži način za pozivanje CORBA metoda.

NAPOMENA

Eto zbog čega postoji interesovanje za konverter IDL-to-Pascal — za kreiranje delova i skeleta koji direktno komunicira sa CORBA ORB-om bez kasnog povezivanja. Borland je nedavno najavio da radi na osnovnoj CORBA podršci, uključujući i CORBA IDL-to-Pascal konverter. Ova podrška se očekuje posle nekog vremena u Delphiju 5 i možda će biti na raspolaganju kada budete čitali ovu knjigu. ■

Uloga CORBA dela je da oponaša udaljeni obejkat u lokalnom adresnom prostoru preuzimanjem poziva, zapisivanjem argumenata u bafer, slanjem na udaljeni server, davanjem rezultata. Na ovaj način program može da funkcioniše kao da je udaljeni objekat lokalan. Uloga skeleta je sasvim suprotna: on se izvršava u adresnom prostoru servera i prihvata zahteve klijenta, raspakujući bafer koji dobija i čineći da server pomisli da radi sa lokalnim klijentom.

Editor biblioteke tipa Vam takođe omogućava da konvertujete SOM IDL kod u CORBA verziju, upotrebom kontrole Export koja se nalazi na samom kraju palete alata. CORBA IDL se može koristiti i u drugim prgoramskim jezicima za generisanje odgovarajućeg interfejsa ili za registrovanje IDL-a u Interface Repositiry Serviceu, kojim upravljaju pomoćni programi `IREP` i `IDL32`. Ovi koraci nisu neophodni za izradu i izvršavanje jednostavnog CORBA primera.

Kada se vratite na primer, onda kada ste ga kompajlirali, možete ga pokrenuti. Nasuprot COM-u, nije potrebno da statički registrujete server, ali ga morate pokrenuti da biste ga učinili dostupnim ORB-u. Pošto program pokušava da se registruje prilikom pokretanja, potrebno je da na sistemu imate instaliran ORB i da se na mreži izvršava Smart Agent pre nego što pokrenete program. CORBA Smart Agent je dinamički, distribuirani servis direktorijuma koji pronalazi raspoloživi server, koji odmah obezbeđuje implementaciju CORBA objekta.

Da biste testirali program na jednom sistemu, jednostavno pokrenite VisiBroker Smart Agenta iz menija Visibroker (koji se nalazi u Borland Delphi 5 elementu menija Windows Start➥Program). U ovom trenutku možete pokrenuti server (ali ne iz Delphi debagera, jer je potrebno da kreirate zasebnu klijent aplikaciju).

Jednostavan CORBA klijent

Dok se server program izvršava, klijent program se može povezati na server. Ukoliko testirate klijenta upotrebom aktivnih podataka u vreme dizajniranja, potrebno je da se server izvršava dok izrađujete klijent program.

Razvoj klijent programa se može nastaviti dodavanjem komponente CorbaConnection formularu. U ovom slučaju, za označavanje odgovarajućeg servera nema combo polja. Potrebno je samo da u svojstvo `RepositoryID` unesete naziv programa i naziv modula podataka odvojene karakterom / (ne tačkom kao kod COM servera). Na primer, možete uneti string *'Corba1/FirstCorba'*. Da biste testirali da li je vrednost dobra i da li sve ispravno funkcioniše, ednostavno promenite vrednost svojstva `Connected` komponente.

U klijent programu možete dodati komponentu `ClientDataSet`, odabrati komponentu `CorbaConnection1` za svojstvo `RemoteServer` i odabrati jedan od interfejsa servera koji su na raspolaganju (ovoga puta ćete upotrebiti combo polje). Na kraju, dodajete komponentu DataSource i neke kontrole koje prepoznaju podatke. Kada kompajlirate i pokrenete program, on će se povezati sa CORBA modulom podataka da bi dobio podatke; to je nešto što možete učiniti i u vreme dizajniranja. Ponoviću, server se mora ručno pokrenuti (što nije bio slučaj sa primerima za COM).

ActiveForm laki klijenti

U ovom poglavlju smo izradili programe lakih klijenata, koji nisu direktno pristupali bazi podataka, već su podatke dobijali od aplikacije servera na takozvanom središnjem čvoru. Neke mreže možda žele da ovaj deo aplikacija prebace na intranet ili Internet, prosleđujući se preko pretraživača. To je ono čemu služi ActiveForms, i to je ono što ćemo učiniti da bismo pokazali program koji ne zahteva nikakvu instalaciju niti konfigurisanje.

Kada bismo izradili ActiveForm koji se povezuje sa MIDAS aplikacijom servera, server bi po potrebi prosledio aktivne SQL podatke, mi bismo mogli da pošaljemo ažurirane podatke nazad i ne bi bilo potrebno da instaliramo BDE na klijent kompjuter.

ActiveForm ovog primera, nazvan AfRemote, povezuje se na jednu od aplikacija servera koju smo izradili ranije u ovom poglavlju (preciznije, na AppServ2). Ukoliko to već niste učinili, potrebno je da izradite i pokrenete ovu aplikaciju servera da biste je registrovali i učinili dostupnom ovoj ActiveX kontroli. Registraciju treba obaviti samo na serveru.

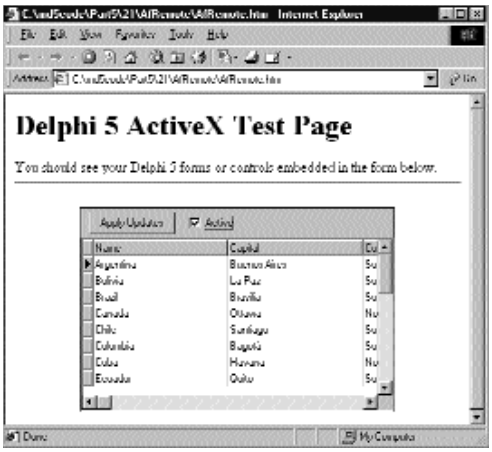
Veza će ovoga puta biti bazirana na komponenti SocketConnection, tako da možemo koristiti TCP/IP; ovo znači da će svaki kompjuter koji ima instaliran Windows 95 i koji je povezan na Internet, moći da je izvršava. Slede ključna svojstva komponenata klijent programa, a to je zapravo ActiveForm:

```
object ActiveRemote: TActiveRemote
  Caption = 'ActiveRemote'
  object DBGrid1: TDBGrid
    Align = alClient
    DataSource = DataSource1
  end
  object Panel1: TPanel
    Align = alTop
    object CheckActive: TCheckBox
      Caption = 'Active'
      OnClick = CheckActiveClick
    end
    object BtnApply: TButton
      Caption = 'Apply Updates'
      OnClick = BtnApplyClick
    end
  end
  object ClientDataSet1: TClientDataSet
    ProviderName = 'DataSetProvider1'
    RemoteServer = SocketConnection1
    OnReconcileError = ClientDataSet1ReconcileError
  end
  object DataSource1: TDataSource
    DataSet = ClientDataSet1
  end
  object SocketConnection1: TSocketConnection
    ServerGUID = '{C5DDE903-2214-11D1-98D0-444553540000}'
    ServerName = 'AppServTwo.RdmCount'
    Address = '127.0.0.1'
  end
end
```

U ovom probnom slučaju, udaljeni kompjuter je naš kompjuter, te sam upotrebio adresu `127.0.0.1`. Ovu adresu zamenite pravom IP adresom koju ima Vaš server. Da bi primer mogao da funkcioniše, server mora da izvršava Borland Socket Server. Kod tri metoda primera AfRemote je prilično direktan:

```
procedure TActiveRemote.CheckActiveClick(Sender: TObject);
begin
  if CheckActive.Checked and not SocketConnection1.Connected then
    SocketConnection1.Connected := True;
  ClientDataSet1.Active := CheckActive.Checked;
end;
procedure TActiveRemote.BtnApplyClick(Sender: TObject);
begin
  if ClientDataSet1.Active then
    ClientDataSet1.ApplyUpdates (-1);
end;
procedure TActiveRemote.ClientDataSet1ReconcileError(
  DataSet: TClientDataSet; E: EReconcileError;
  UpdateKind: TUpdateKind; var Action: TReconcileAction);
begin
  Action := HandleReconcileError (DataSet, UpdateKind, E);
end;
```

Kao i ranije, greška ažuriranja će prikazati standardni okvir za dijalog Reconcile. Izlaz programa AfRemote u Microsoftovom Internet Exploreru je prikazan na slici 21.12.



SLIKA 21.12 *ActiveForm primera AfRemote koristi aktivne podatke koje dobija od aplikacije servera i poštuje Delphijevu višelinijsku arhitekturu*

Mada sasvim sigurno možete upotrebiti ActiveForm kao MIDAS klijenta, ovaj pristup ima nekoliko problema. Prvo, korisnici moraju imati Win32 kompjuter i Internet Explorer (ne neki drugi pretraživač) i to verovatno najnoviju verziju. Drugo, preuzimanje ActiveX komponenata korisnika izlaže korisnika riziku koji nije zanemarljiv, baš kao i izvršavanje programa koji je preuzet sa Weba. Ja ovde ne govorim samo o virusima koje ActiveX komponente mogu ubaciti u Vaš

kompjuter, već na činjenicu da ActiveX kontrola može uzeti bilo koji fajl sa Vašeg kompjutera i poslati ga serveru, a da Vi to ne znate. U praksi, mnogi korisnici Weba onemogućavaju ActiveX podršku, čak i u pretraživačima koji je omogućavaju. Na kraju, preuzimanje ActiveX kontrole može zahtevati dosta vremena preko spore Internet veze.

Kao što sam ranije istakao, ovaj pristup ima smisla na lokalnoj mreži, ali ne i u otvorenom svetu Interneta, naročito ukoliko želite da Vaš web sajt poseti što je moguće više korisnika. Alternativni pristup koji uvodi Delphi 5 je Web MIDAS klijent, koji je moguć upotrebom tehnologije koja se naziva Internet Express.

Internet Express

Sada kada znamo kako da izradimo MIDAS severe i klijent programe, možda želimo da otvorimo ovu arhitekturu i generišemo HTML strane na Webu da bismo omogućili svakom korisniku da komunicira sa našim serverom u središnjem čvoru preko web servera. Ideja koja stoji iza Internet Expressa je da Vi pišete web server proširenje (CGI ili ISAPI, kao što smo razmatrali u prethodnom poglavlju), koje proizvodi web strane koje su povezane sa našim MIDAS serverom. Vaše klijent aplikacije se ponašaju kao MIDAS klijent i proizvode strane za pretraživač klijenta. Internet Express nudi servise koji su neophodni za laku izradu ovakve aplikacije.

Ja znam da ovo zvuči zbunjujuće, ali Internet Express je višelinijska (zapravo postoje četiri linije) arhitektura: SQL server, server aplikacije (MIDAS server), web server sa aplikacijom i, na kraju, web pretraživač. Naravno, prva tri nivoa možete smestiti na isti kompjuter, ali još uvek će postojati logička podela na četiri nivoa. Takođe, možete zaobići MIDAS nivo povezujući web server sa lokalnim fajlom.

Internet Express koristi više tehnologija da bi ovo postigao:

- MIDAS paketi podataka (koji su zasnovani na OleVariantima u Delphi implementaciji) se konvertuju u XML format da bi programu omogućili da podatke umetne na HTML strane. Zapravo, Delta paket podataka je takođe predstavljen u XML-u. Ove operacije obavlja nova komponenta XMLBroker, skup podataka koji je veoma sličan komponenti ClientDataSet, koja može rukovati XML-om i obezbediti podatke za nove JavaScript komponente.
- Postoji nova komponenta MidasPageProducer koja Vam omogućava da generišete HTML formulare na osnovu skupova podataka, na vizuelan način sličan izradi Delphi formulara. Umesto upotrebe VCL komponenata, koristite JavaScript komponente.
- Da bi operacije izmena na klijent strani bile moćne, komponenta MidasPageProducer koristi specijalne JavaScript komponente i JavaScript kod. Delphi 5 sadrži prilično veliku JavaScript biblioteku koju pretraživači moraju da preuzmu. Ovo Vam, možda, izgleda problematično, ali to je jedini način da interfejs pretraživača (koji se zasniva na dinamičkom HTML-u) bude dovoljno bogat da podrži veze polja i druga pravila. Ovo je zaista nemoguće kada se upotrebljava samo običan HTML.

Naravno, da biste prosledili ovakvu arhitekturu, nije potrebno ništa posebno na klijent strani, jer se bilo koji pretraživač do standarda HTML 4 (koji mnoge pretraživače izbacuje iz igre) može koristiti na bilo kom operativnom sistemu! Web server, umesto toga, mora biti Win32 server i mora proslediti MIDAS (pošto platite odgovarajuću licencu, čak i kada web server povežete sa lokalnim fajlom).

Izrada prvog primera

Moj prvi Internet Express primer, koji sam nazvao IeFirst, veoma je jednostavan i uključuje samo minimalan broj elemenata koji su potrebni da se jednostavni MIDAS klijent (u ovom slučaju ThinCli1) pretvori u interfejs na bazi pretraživača. Ja sam kreirao novu CGI aplikaciju i dodao joj DCOMConnection komponentu povezanu sa AppServ1 serverom. Naredni korak je dodavanje XMLBroker komponente i njeno povezivanje sa udaljenim serverom i provajderom:

```
object XMLBroker1: TXMLBroker
  ProviderName = 'DataSetProvider1'
  RemoteServer = DCOMConnection1
  WebDispatch.MethodType = mtAny
  WebDispatch.PathInfo = 'XMLBroker1'
  ReconcileProducer = PageProducer1
  OnGetResponse = XMLBroker1GetResponse
end
```

Komponenta `ReconcileProducer` je neophodna da bi se prikazala odgovarajuća poruka o grešci u slučaju konflikta ažuriranja. Kao što ćemo kasnije videti, jedan od Delphijevih demoa sadrži korisnički kod, ali u ovom jednostavnom primeru sam samo povezao PageProducer sa generičkom HTML porukom o grešci. Posle podešavanja XML brokera možete dodati komponentu MidasPageProducer web modulu podataka. Ova komponenta sadrži standardni HTML skelet koji možete prilagoditi bez upotrebe specijalnih elemenata:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<#INCLUDES><#STYLES><#WARNINGS><#FORMS><#SCRIPT>
</BODY>
</HTML>
```

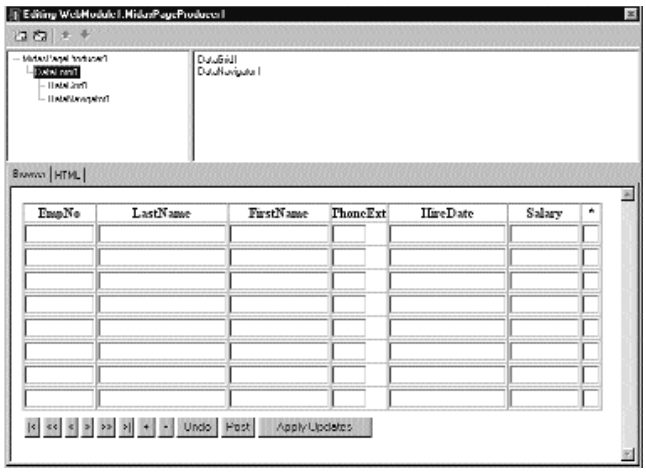
Specijalni tagovi se automatski proširuju upotrebom JavaScript fajlova direktorijuma koji je naznačen svojstvom `InclidePathURL`. Ovo svojstvo morate podesiti tako da se referiše na direktorijum web servera u kojem se nalaze ovi fajlovi. Možete ih pronaći u poddirektorijumu Source/Webmidas direktorijuma Delphi 5.

SAVET

U komponenti MidasPageProducer možete se referisati na spoljašnji fajl stila ili na ugnežđeni skup stilova. Podrška stilovima u ovoj komponenti i u celoj Internet Express arhitekturi je prilično kompletna. ■

Da biste prilagodili rezultujući HTML komponente MidasPageProducer, možete upotrebiti njen editor, a to je prilično složen alat. Samo dva puta kliknite komponentu, a Delphi će otvoriti prozor kao što je prozor prikazan na slici 21.13. U ovom editoru možete kreirati složene strukture, počevši od

formulara upita, formulara podataka ili generičke grupe. Formularu podataka mog primera sam dodao komponente DataGrid i DataNavigator, a da ih nisam prilagođavao (operacija kojom dodajete dete kontrole, kolone i druge objekte koji zamenjuju unapred određene objekte).



SLIKA 21.13 *MidasPageProducer editor Vam omogućava da vizuelno izradite složene HTML formulare*

DFM kod ovih komponenata u mom primeru izgleda ovako:

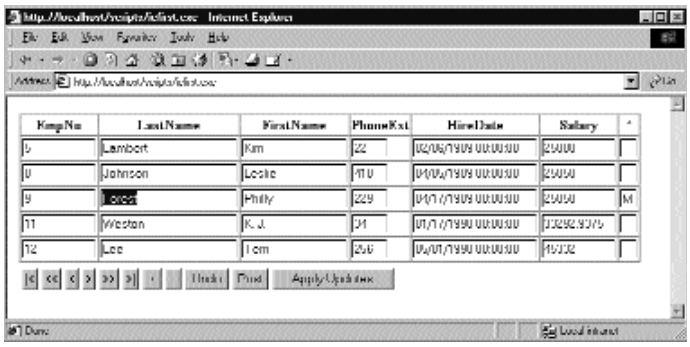
```
object DataForm1: TDataForm
  object DataGrid1: TDataGrid
    XMLBroker = XMLBroker1
    DisplayRows = 5
    TableAttributes.CellSpacing = 0
  end
  object DataNavigator1: TDataNavigator
    XMLComponent = DataGrid1
  end
end
```

Međutim, vrednost ovih komponenata je u HTML kodu koji proizvode, i koji možete pogledati ukoliko odaberete HTML karticu MidasPageProducer editora. Početni deo definicije tabele u ovom HTML skriptu izgleda ovako (ovde je vidljiva samo jedna ćelija podataka):

```
<FORM NAME=DataForm1>
<TABLE><TR><TD COLSPAN=2>
<TABLE CELLSPACING=0 BORDER=1><TR>
<TH>EmpNo</TH>
<TH>LastName</TH>
...
</TR><TR><TD><DIV><INPUT TYPE=TEXT
  NAME="DataGrid1_EmpNo"
  SIZE=10
  onFocus='if(xml_ready)DataGrid1_Disp.xfocus(this);'
  onkeydown='if(xml_ready)DataGrid1_Disp.keys(this);'>
</DIV></TD>
...
```

Kada je HTML generator podešen, možete se vratiti na web modul podataka, dodati mu akciju i povezati akcije sa komponentom MidasPageProducer preko svojstva `Producer`. Ovo bi trebalo da bude dovoljno da bi program funkcionisao preko pretraživača, kao što možete videti na slici 21.14. Ukoliko pogledate HTML fajl koji je primio pretraživač, videćete definiciju tabele koja je ranije pomenuta, JavaScritp kod tu i tamo, i podatke baze podataka u XML formatu (koji su predstavljeni kao metapodaci):

```
<XML ID=XMLBroker1_Doc>
  <DATAPACKET Version="2.0">
    <METADATA>
      <FIELDS>
        <FIELD attrname="EmpNo" fieldtype="i4"/>
        <FIELD attrname="LastName" fieldtype="string"
          WIDTH="20"/>
        ...
      </FIELDS>
      <PARAMS DEFAULT_ORDER="1"
        PRIMARY_KEY="1" LCID="1033"/>
    </METADATA>
    <ROWDATA>
      <ROW EmpNo="2" LastName="Nelson" FirstName="Robert"
        PhoneExt="250" HireDate="19881228" Salary="40000"/>
      <ROW EmpNo="4" LastName="Young" FirstName="Bruce"
        PhoneExt="233" HireDate="19881228" Salary="55500"/>
```



SLIKA 21.14 *Primer IeFirst u web pretraživaču. Obratite pažnju na M kod u poslednjoj koloni koji označava da je slog izmenjen*

Ove podatke je obezbedio XML broker i prosledio ih je komponenti Producer koja se nalazi u HTML fajlu. Primetićete da broj slogova koji se šalje klijentu zavisi od XMLBrokera, ne od broja linija tabele. Kada se XML podaci pošalju pretraživaču, možete koristiti kontrole komponente navigatora za pregled podataka, a da ne zahtevate dalji pristup serveru radi dobijanja novih podataka.

Istovremeno, klase JavaScripta u sistemu omogućavaju korisniku da unese nove podatke, poštujući pravila koja nameće JavaScript kod koji je povezan sa dinamičkim komponentama. Primetićete da tabela sadrži dodatnu asterisk kolonu kojom se označavaju slogovi koji su izmenjeni. Ažurirani podaci se dobjiaju XML paketom podataka u pretraživaču, a šalju se nazad

serveru kada korisnik klikne kontrolu Apply Updates. U tom trenutku pretraživač aktivira akciju naznačenu svojstvom `WebDispatch.PathInfo` komponente XMLBroker. Nema potrebe izvoziti ovu akciju iz web modula podataka jer je ova operacija automatska (mada je možete onemogućiti određivanjem vrednosti `False` za svojstvo `WebDispatch.Enable`).

XML komponenta primenjuje izmene na server, vraćajući sadržaj provajdera koji je povezan sa svojstvom `ReconcileProvider` (ili pozivajući se na izuzetak ukoliko nije definsano). Kada sve lepo funkcioniše, XMLBroker komponenta preusmerava kontrolu na glavnu stranu koja sadrži podatke. Ipak, ja sam iskusio neke probleme upotrebljavajući ovu tehniku kada sam koristio Personal Web Server za Windows 98, te zbog toga primer IeFirst obrađuje događaje `OnGetResponse` sledećim kodom:

```
procedure TWebModule1.XMLBroker1GetResponse(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse;
  var Handled: Boolean);
begin
  Response.Content := '<h1>Updated</h1><p>' +
    MidasPageProducer1.Content;
  Handled := True;
end;
```

Master/detail na Webu

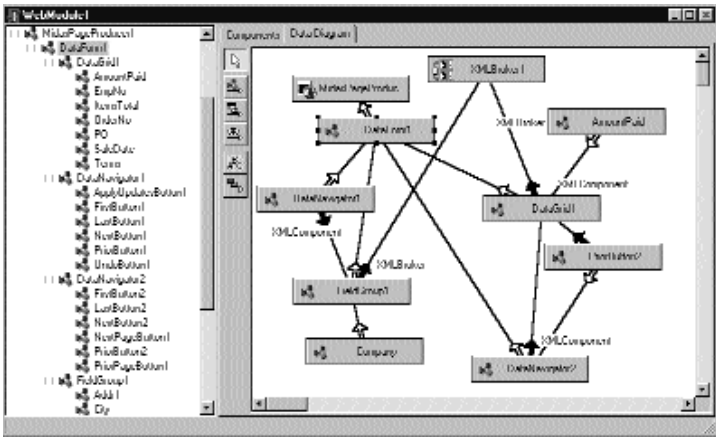
Moj drugi i poslednji Internet Express primer daje nešto više od osnovnog master/detail paketa podataka za web pretraživanje. Program koristi AppSPlus server, koji definiše master/detail zavisnost. Polje skupa podataka koje je umetnuto u tabelu će biti transformisano u XML strukturu, a prikazivaće iste podatke.

Program koristi kombinaciju komponenata XMLBroker, MidasPageProducer i DCOMConnection kao u prethodnom primeru. Ovoga puta sam prilagodio web komponente, kreirajući polja i birajući informacije koje će biti prikazane. Deo ove strukture možete videti u pogledu Tree web modula podataka na slici 21.15, kao i složene veze između nekih komponenata u pogledu Data Diagram. Sledi prilično dugačak listing ove strukture: uklonio sam neke dodatne informacije, ali smatram da ga vredi pogledati:

```
object XMLBroker1: TXMLBroker
  ProviderName = 'ProviderCustomer'
  RemoteServer = DCOMConnection1
  WebDispatch.PathInfo = 'XMLBroker1'
end
object MidasPageProducer1: TMidasPageProducer
  IncludePathURL =
    'C:/Program Files/Borland/Delphi5/Source/Webmidas/'
  object DataForm1: TDataForm
    object DataNavigator1: TDataNavigator
      XMLComponent = FieldGroup1
      object FirstButton1: TFirstButton
        XMLComponent = FieldGroup1
        Caption = '|<'
      end
      object PriorButton1: TPriorButton
        XMLComponent = FieldGroup1
```

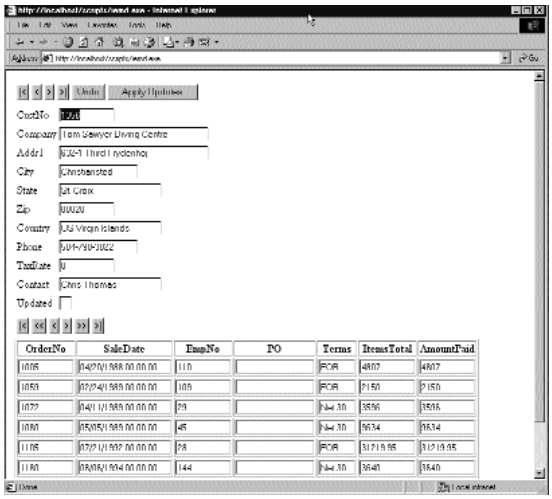
```
    Caption = '<'
  end
  object NextButton1: TNextButton
    XMLComponent = FieldGroup1
    Caption = '>'
  end
  ...
  object ApplyUpdatesButton1: TApplyUpdatesButton
    Caption = 'Apply Updates'
    XMLBroker = XMLBroker1
    XMLUseParent = True
  end
end
object FieldGroup1: TFieldGroup
  XMLBroker = XMLBroker1
  object CustNo: TFieldText
    DisplayWidth = 10
    Caption = 'CustNo'
    FieldName = 'CustNo'
  end
  object Company: TFieldText
    DisplayWidth = 30
    Caption = 'Company'
    FieldName = 'Company'
  end
  ...
end
object DataNavigator2: TDataNavigator
  XMLComponent = DataGrid1
  object FirstButton2: TFirstButton
    XMLComponent = DataGrid1
    Caption = '¦<'
  end
  object PriorPageButton1: TPriorPageButton
    XMLComponent = DataGrid1
    Caption = '<<'
  end
  ...
end
object DataGrid1: TDataGrid
  XMLBroker = XMLBroker1
  XMLDataSetField = 'TableOrders'
  DisplayRows = 8
  object OrderNo: TTextColumn
    DisplayWidth = 10
    Caption = 'OrderNo'
    FieldName = 'OrderNo'
  end
  object SaleDate: TTextColumn
    DisplayWidth = 18
    Caption = 'SaleDate'
    FieldName = 'SaleDate'
  end
  ...
```

```
    end
  end
end
object DCOMConnection1: TDCOMConnection
  Connected = True
  ServerName = 'AppSPlus.AppServerPlus'
end
```



SLIKA 21.15 *Struktura komponente MidasPageProducer primera IeMd*

Kada je struktura podešena, možete proslediti CGI izvršni fajl na web server i dobićete efekat prikazan na slici 21.16 direktno u web pretraživaču. Primetićete da je HTML koji dobijate prilično dugačak jer sadrži celu master/detail strukturu. Kada ga dobijete, možete pretraživati master tabelu i tabelu sa detaljima, a da od servera ne tražite podatke.



SLIKA 21.16 *Master/detail zavisnosti prikazane u pretraživaču izvršavanjem primera IeMd*

Očigledno, može se još mnogo toga reći o mogućnostima Internet Express MIDAS komponenata Delphija 5 i o tehnologijama koje to omogućavaju, jer Vam, zapravo, nisam predstavio XML i JavaScript. Moj cilj je bio samo da Vam dam ideju šta se može učiniti i koliko brzo se to može postići upotrebom ove potpuno nove arhitekture Delphija 5, koja mnogo obećava u oblasti web programiranja i koja brzo napreduje.

Šta je sledeće?

Borland/Inprise je podršku za pravu višelinijsku arhitekturu prvi put predstavio u Delphiju 3, a proširio ju je u Delphiju 4 i Delphiju 5 da bi podržao TCP/IP priključke, MTS, CORBA i HTTP, pa čak i MIDAS klijente web pretraživača. Kompanija stalno proširuje ovu arhitekturu da bi igrala osnovnu ulogu u budućem klijent/server programiranju. Takođe se obraća velika pažnja na CORBA podršku koju obezbeđuje Visigenic ORB, i verovatno ćemo uskoro imati i IDL-to-Pascal mapiranje. Istovremeno, ugovor koji je Inprise potpisao sa Microsoftom u proleće 1999. godine, obezbeđuje solidne osnove za najbolje razvojne alate za COM, DCOM i MTS (i COM+ u bliskoj budućnosti).

Naravno, neću se mnogo upuštati u ove teme, ali sam mislio da ih vredi pomenuti na kraju ove knjige, dok pokušavam da Vam dam nekoliko saveta o tome šta je sledeće za Delphi programere. Delphi se svakako ne može zanemariti — kako na tržištu Windowsa tako i na tržištima server/klijent aplikacija i aplikacija projekata — i verovatno je najbolji alat ukoliko Vam je u oba smera potrebna kontrola i snaga. Sada Delphi 5 obezbeđuje i kompletnu platformu za web programiranje.

Jednako kao što Borland želi da obezbedi najbolje alate za programere, ja se nadam da Vam je ova knjiga pomogla da savladate Delphi, najuspešniji alat koji je Borland izneo na tržište u proteklih nekoliko godina. Ukoliko želite da se udubite u tajne VCL biblioteke na kojoj je zasnovan Delphi, pokušajte sa knjigom "Delphi developer's Handbook" (Sybex), u kojoj sam koautor sa Timom Gučom (Tim Gooch) i Džonom Lamom (John Lam). Takođe, proverite reference i napredni materijal koji sam prikupio na svom web sajtu (`www.marcocantu.com`). Ovaj materijal nije mogao da bude uključen u knjigu samo zbog ograničenog prostora.

Takođe, postoji i dodatno poglavlje koje razmatra grafiku u Delphiju, a koje možete pronaći na Sybexovom web sajtu (`www.sybex.com`). Preuzmite ga ukoliko to već niste učinili i pratite oba web sajta za moguća nova izdanja ove knjige.

Indeks

A

B

C

Č

Ć

D

E

F

G

H

I

J

K

L

M

N

O

P

Š

T

U

V

W

Y

Z