Delfinárium

Prvá časť: Úvod

Svojho času vzbudilo Delphi vo svete programátorov nemalý rozruch. Vývojári konečne dostali k dispozícii nástroj, ktorý im umožnil kráčať s dobou. Tvorcovia Delphi sa do značnej miery inšpirovali Visual Basicom (ďalej VB), ale svoj nový vývojársky nástroj postavili na vyspelejšom programovacom jazyku – jazyku Pascal. Na prvý pohľad sa Delphi od VB príliš nelíši; oba nástroje poskytujú vynikajúce prostriedky na návrh aplikácií. Väčšie rozdiely zbadáme, ak sa na obe vývojové prostredia pozrieme „pod lupou".

VB, hoci je jednoduchý a výkonný, nevytvára skutočné (native) EXE súbory. Kód, ktorý vytvorí, síce má príponu EXE, vyžaduje však prítomnosť knižnice VBRUNxx.DLL (xx je číslo verzie), bez ktorej nedokáže pracovať. Naproti tomu Delphi vytvára skutočné EXE súbory, a teda nie sú potrebné nijaké run-time knižnice. EXE súbory vytvorené VB sa vzdialene podobajú akýmsi makrám či skriptom, ktoré knižnica VBRUNxx.DLL vykonáva (interpretuje), čo sa odrazí aj na výkone, pretože interpretovaný kód je pomalší ako natívny kód. Faktom však zostáva, že VB je aj napriek tomu obľúbený nielen u začiatočníkov, ale aj u profesionálnych vývojárov.

A v čom boli tieto nástroje také revolučné? Predovšetkým v novom chápaní pojmu vývoj aplikácie. Kedysi dávno (pre rokom 1990) programátor dostal úlohu, sadol si a začal písať program. Zadaná úloha bola jednoduchá, niekedy bol program hotový už na druhý deň. S príchodom Windows však nároky na schopnosti vývojára podstatne vzrástli. C++, ktoré rozšírilo svoje impérium aj do sveta „okien", bolo prakticky jediným uznávaným programovacím jazykom. Kto nemal detailné znalosti Windows API, prakticky nemal šancu. Aj jednoduchý program typu Hello world vyžadoval mnoho trpezlivosti a značné úsilie. Odpoveďou na bolestné kvílenie utrápených vývojárov bol VB. Hoci nebol taký efektívny ako C++, dal sa omnoho rýchlejšie naučiť a vývoj aplikácií si vyžadoval iba zlomok času oproti C++. Vo firme Inprise (predtým Borland) však tiež nezaspali na vavrínoch, a tak sa aréna vývojárskych nástrojov pre Windows stala bohatšou o jedného bojovníka – Delphi. V súvislosti s Delphi sa často skloňuje termín RAD. Táto záhadná skratka značí Rapid Application Development, čo znamená rýchly vývoj aplikácií. Časy, keď sa človek musel predierať hustým pralesom Windows API funkcií, sú naveky preč. V dnešnej dobe programátor „skladá" svoje programy z komponentov, akýchsi základných stavebných „kameňov" každej aplikácie. Ak potrebujeme pracovať s databázami, nemusíme prácne vytvárať vlastné rutiny, jednoducho použijeme vhodný komponent. Niekedy je takto možné „poskladať" celý program aj bez jediného riadka kódu.

OOP

OOP znamená Object Oriented Programming. Objekty sú azda najzákladnejším prvkom Delphi. Celá VCL (Visual Component Library, knižnica komponentov dodávaná s Delphi) je objektovo založená. Všetky komponenty sú vlastne objektmi. Objekty však nie sú komponentmi. Vo všeobecnosti sa pod pojmom komponent rozumie objekt, ktorý bol odvodený od triedy TComponent. Pre mnohých začiatočníkov je tematika objektov komplikovaná, preto sa budem snažiť problém objasniť čo najjednoduchšie. Dúfam preto, že profesionáli mi určité zjednodušenia a nepresnosti odpustia. Každý objekt sa skladá z dvoch hlavných častí: dát a kódu. Do kódovej časti patria metódy, ktoré majú podobu procedúr alebo funkcií. Do dátovej časti objektu patria vlastnosti (properties), ktorých nastavovanie ovplyvňuje správanie objektu. Veľkou výhodou pri používaní objektov je ich flexibilita. Ak chceme vytvoriť objekt podobný už existujúcemu, hotový objekt jednoducho použijeme ako „predlohu" pre náš nový objekt. Ak napríklad chceme vytvoriť pokročilejšiu verziu komponentu TLabel, nemusíme začať od nuly. Jednoducho použijeme TLabel ako „predlohu" a náš komponent po ňom zdedí všetky vlastnosti a metódy. Potom pridáme vlastné rutiny a veldielo je hotové. V súvislosti s komponentmi spomeniem a vysvetlím pojem udalosť (event). Udalosť je vlastne dôsledok interakcie programu s používateľom. Jednoducho povedané, program reaguje na určité podnety. Ako príklad spomeniem udalosť OnClick, ktorá vznikne po kliknutí myšou, či udalosť OnMouseMove, ktorá je generovaná pri pohybe myši. Aby sme vedeli na príslušné udalosti zareagovať, musíme pre ne napísať obslužné rutiny, tzv. event handlery. Ukážeme si jednoduchý príklad: vytvoríme formulár a pridáme jedno tlačidlo. Obsluha udalosti OnClick bude vyzerať takto:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
     ShowMessage('Hello world!')
end;
```

Program spustíme a otvoríme šampanské. Naša prvá Delphi aplikácia je na svete.

Výnimky

Veľmi dôležitým prvkom jazyka ObjectPascal sú výnimky, ktoré značne zjednodušujú obsluhu chybových stavov. V starom dobrom Turbo Pascale sme testovali úspešnosť operácií I/O prostredníctvom funkcie IOResult. Ak bola jej hodnota nulová, operácia prebehla úspešne, v opačnom prípade sa niekde stala chyba. Keby neexistovali výnimky, museli by sme podobným spôsobom testovať úspešnosť všetkých operácií. ObjectPascal nás od podobných situácií úspešne odbremení. Princíp výnimiek je jednoduchý, základom všetkého sú bloky `try...except` a `try..finally`. Pokúsme sa napríklad otvorit tabulku:

```
try
          MyTable.Open;
except
          ShowMessage(,Chyba pri otváraní');
end;
```

Ak operácia MyTable.Open neuspeje, program zacne vykonávat blok except...end, teda zobrazí chybové hlásenie. Výnimky sú objektmi odvodenými od základného objektu Exception. Dôležitou vlastnosťou tohto objektu je vlastnosť Message, ktorá je typu string a obsahuje text chybového hlásenia, ktoré sa pri vzniku výnimky zobrazí. Náš príklad teraz trochu vylepšíme. Do vlastnosti TableName komponentu TTable napíšeme názov neexistujúcej tabuľky. Náš kód zachytí chybové hlásenie „Table does not exist " a vypíše jeho slovenskú verziu.

```
try
    table1.Open;
except
   on E:Exception do
   begin
   if pos('Table does not exist',E.Message)>0 then ShowMessage('Tabulka
neexistuje.')
               else raise;
   end;
   end;
```

Všimnite si použitie funkcie Pos. Táto funkcia nám vracia index prvého písmena podreťazca v retazci písmen. Inými slovami, náš príklad skúma, či dané chybové hlásenie obsahuje reťazec „Table does not exist". Použil som túto funkciu preto, aby sme nemuseli používať text celého chybového hlásenia, ktorý je v skutočnosti trochu dlhší a obsahuje aj názov tabuľky, ktorú sa nepodarilo otvoriť. Ak sa nechceme zaťažovať analýzou všetkých chybových správ, môžeme jednoducho použiť nasledujúce riešenie:

```
try
  table1.Open;
except
  on EDatabaseError do ShowMessage('Chyba pri práci s databázou')
  else raise;
end;
```

Po preštudovaní oboch príkladov si mnohí z vás položia otázku, čo znamená to záhadné else raise. Príkaz raise vyvolá tzv. implicitnú obsluhu výnimky (default exception handler). Ak nemáme v úmysle prekladať anglické chybové správy do slovenčiny, použijeme v bloku except...end príkaz raise a Delphi zobrazí implicitné chybové hlásenie. V našom prípade teda program najprv zistí, či ide o chybu databázového typu, a ak áno, zobrazí hlásenie „Chyba pri práci s databázou". Ak výnimka vznikne pre niečo iné (napr. pre neexistenciu určitého adresára, v ktorom sa tabuľka mala nachádzať), riadenie preberá implicitná obsluha výnimky, ktorá zobrazí príslušné chybové hlásenie. Čo si však počneme v prípade, keď sme napríklad vyhradili pamäť pre obrázok, zrazu sa však nečakane vyskytla výnimka a my potrebujeme vyhradenú pamäť uvoľniť? Riešením nášho problému je použitie bloku try...finally. Hlavný rozdiel oproti try...except spočíva v tom, že kód finally ...end sa vykoná bez ohľadu na to, či sa v bloku try...finally vyskytla chyba, alebo nie. Nasledujúci príklad nám daný problém demonštruje v praxi:

```
try
    BitMap1 := TBitMap.Create;
    BitMap1.LoadFromResourceName(HInstance,'THEBITMAP');
    Canvas.Draw(12,12,BitMap1);
finally
    BitMap1.Free;
end;
```

Po spustení tohto programu dostaneme chybové hlásenie „Resource THEBITMAP not found". Pamäť, ktorú sme pre bitmapu vyhradili, sa však bez problémov uvoľní nezávisle od toho, či bola operácia v bloku try...finally úspešná, alebo nie. Dúfam, že vás môj takmer čisto teoretický výklad príliš nenudil. Nabudúce si ukážeme, ako vyzerá uvedená teória v praxi.

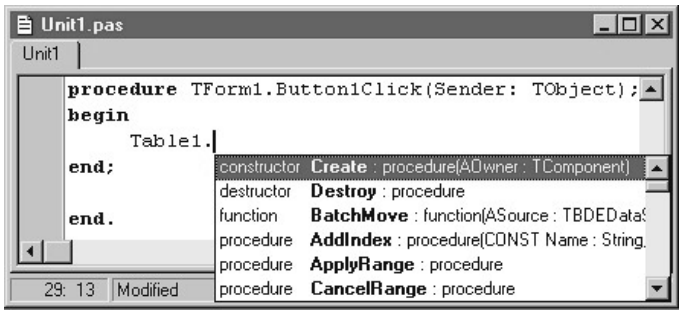
Druhá časť: Prvá miniaplikácia

V predošlej časti seriálu sme rozobrali niektoré dôležité pojmy ako napr. OOP, vlastnosti (properties), metódy (methods), udalosti (events) a výnimky (exceptions). Najprv si však povieme niečo o Object Inspectore, nástrojoch Code Insight a potom si ukážeme, ako to všetko funguje v praxi.

Object Inspector

Object Inspector je najpoužívanejší nástroj Delphi. Pomocou neho je možné nastavovať vlastnosti komponentu (na karte Properties) a vytvárať obslužné rutiny udalostí (dvojitým kliknutím vedľa názvu udalosti na karte Events). Tradičný postup, používaný v starších „nevizuálnych" vývojových nástrojoch, bol asi taký, že ste program museli aj po najmenšom zásahu rekompilovať. V praxi to vyzeralo asi takto: „Tak, teraz sem dám jedno tlačidlo." (Po umiestnení tlačidla nasleduje jednominútová rekompilácia.) „No, to tlačidlo by malo byť trochu viac vlavo," (nasleduje ďalšia jednominútová rekompilácia) „nie, predsa len malo zostať tam, kde bolo," (ďalšia rekompilácia). V Delphi vytvoríte komplexný návrh vzhľadu aplikácie bez jedinej kompilácie či spustenia.

Použitie Object Inspectora je jednoduché: myšou vyberieme príslušný komponent a Object Inspector zobrazí vlastnosti, s ktorými je možné manipulovať (niektoré vlastnosti možno meniť len počas behu programu, tie sa v Object Inspectore neobjavia). Len čo sme s nastaveniami spokojní, klikneme na kartu Events (v hornej časti Object Inspectoru) a môžeme vytvárať obslužné rutiny pre rôzne typy udalostí. Veľkým prínosom je, že Delphi vygeneruje niektoré „formality" týkajúce sa obslužných rutín za nás, nemusíme teda prácne vypisovať konštrukcie typu Procedure TForm1.Button1Click(Sender: TObject).



Obr. 1

Nástroje Code Insight

Šťastným majiteľom Delphi 3 a 4 písanie kódu značne urýchlia nástroje Code Insight, ktoré je možné aktivovať z karty Code Insight (menu Tools – Environment Options).

Zaškrtnutím Code Completion máte po stlacení kombinácie Ctrl + medzerník k dispozícii zoznam vlastností, udalostí a metód danej triedy (obr. 1). Nemusíte preto zdlhavo vypisovať Table1.FieldByName(....), stačí, keď napíšete Table1. a stlačíte uvedenú kombináciu klávesov. Zo zoznamu potom vyberiete vlastnosť FieldByName. Zaškrtnutie voľby Code Parameters spôsobí, že po dopísaní názvu metódy ukončeného zátvorkou sa budú v bublinovej nápovedi postupne zobrazovať názvy a druh parametrov (obr. 2).

Tooltip Expression Evaluator nám umožní sledovať stav premenných bez nutnosti ich zadávania do okna Watches. Stačí zastaviť program, ukázať kurzorom na určitú premennú a v bublinovej nápovedi sa nám zobrazí jej obsah.

Teória v praxi

Ako som minule spomínal, vlastnosti patria do dátovej časti objektu. Zmenou vlastnosti na inú hodnotu sa mení správanie objektu, napr. jeho farba či iné atribúty. Tieto zmeny sú obzvlášť nápadné najmä pri vizuálnych komponentoch. Azda najčastejšie používanými vlastnosťami sú Name a Caption. Name je vlastne názov komponentu a Caption je reťazec, ktorý bude komponent zobrazovať (napr. text v titulnom pruhu formulára, text tlačidla atd.). Tu treba dodať, že vo vlastnosti Name nie je možné (na rozdiel od Caption) používať diakritiku ani medzery.

Spustíme teda Delphi a z menu New vyberieme New Application. Na formulár umiestnime komponent TPanel, nastavíme vlastnosť Align na hodnotu alBottom a vyma-

Obr. 2

Obr. 3

žeme reťazec „Panel1" z vlastnosti Caption. Potom na panel pridáme tri tlačidlá s nasledujúcim popisom (caption): Žltá, Modrá, Sivá. Aby bol náš program prehľadnejší, nemali by sme nechávať komponentom mená, ktoré im priradilo Delphi (teda Button1, Button2, atd). Preto som dal tlacidlám názvy žlta, modrá a šeda (teda podľa farby, ktorú reprezentujú).

Obsluha udalosti OnClick tlačidla žlta bude vyzerať takto:

```
procedure TForm1.zltaClick(Sender: TObject);
begin
     form1.Color:=clYellow;
end;
```

Podobným spôsobom zmeníme farbu formulára na modrú:

```
form1.Color:=clBlue;
a na sivú:
form1.Color:=clGray;
```

Náš program by mal po stlačení príslušného tlačidla zmeniť pozadie formulára. Aby sme zistili, že k stlačeniu vôbec došlo, potrebujeme sa skamarátiť s udalostami. Ako som minule spomínal, udalosti sú dôsledkom interakcie používateľa s programom. Každý komponent ich má niekoľko, spomeniem napríklad udalosť OnClick (má ju takmer každý vizuálny komponent), ktorá sa vyvolá po stlačení ľavého tlačidla myši, OnMouseMove, ktorá sa vyvolá pri pohybe myši nad formulárom, OnDblClick, ktorá sa vyvolá pri „double clicku" (teda dvojitom kliknutí) na komponent. Treba pripomenúť, že udalosti samy osebe nič nerobia, ich cieľom je vyvolať obslužnú rutinu a odovzdať jej potrebné údaje v podobe parametrov. Celý kód programu vlastne pozostáva z obslužných rutín rôznych udalostí. Teóriu si teraz ukážeme na príklade. Do obsluhy OnMouseDown nášho formulára pridáme nasledujúci kód:

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
     Canvas.TextOut(x,y,'Hello world!');
end;
```

Výsledok vidíme na obrázku č. 3. Aby program mohol zobraziť správu na miesto, nad ktorým sme klikli myšou, potrebuje vedieť presné súradnice, na ktorých ku kliknutiu došlo. Tieto súradnice získame pomocou parametrov, ktoré naša obslužná rutina „dostala" pri vyvolaní udalosti. Okrem parametrov X a Y máme ešte k dispozícii parameter Button a parameter Shift, ktorý nás oboznámi so stavom klávesov Alt, Shift a Control. Náš príklad teraz trochu modifikujeme:

```
if ssCtrl in Shift then Canvas.TextOut(x,y,'Control stlacený') else
canvas.TextOut(x,y,'Hello world');
```

Ak klikneme na formulár a pridržíme kláves Control, vypíše sa správa „Control stlačený", v opačnom prípade program vypíše iba „Hello world".

Stručné vysvetlenie vlastností a udalostí máme teda za sebou, ostáva ešte objasniť problematiku metód. Metódy sú vlastne „výkonnou časťou" objektu. Znamená to, že zavolaním metódy objekt vykoná nejakú činnosť, ktorá bude ovplyvnená nastavením vlastností. Za zmienku stojí, že niekedy netreba priamo volať metódu, pretože tá môže byť volaná aj v rámci zmeny hodnoty niektorej vlastnosti na inú hodnotu. Tak napríklad databázovú tabuľku môžeme otvoriť dvoma spôsobmi: volaním metódy Open alebo nastavením vlastnosti Active komponentu TTable na true. Metódy majú často formu funkcií, pretože neraz je potrebné používateľa informovať o výsledku operácie. Napríklad na vyhľadávanie v databáze sa používa metóda Locate komponentu TTable. Ak sa podarilo nájsť záznam spĺňajúci potrebné kritériá, metóda vráti hodnotu true, inak vráti hodnotu false. My si teraz ukážeme využitie metódy Minimize triedy TApplication, ktorá po stlačení tlačidla „Minimalizovať" minimalizuje formulár.

```
procedure TForm1.MiniClick(Sender: TObject);
begin
     Application.Minimize;
end;
```

Náš ukážkový program je síce pekný, no prakticky nepoužiteľný. Preto si niektoré črty OOP ukážeme ešte raz, tentoraz však už na zmysluplnejšom programe.

Jednoduchý editor

Spustíme Delphi a z menu File vyberieme New Application. Potom trochu zmenšíme formulár a vlastnosť Position zmeníme na poScreenCenter, čím zabezpečíme, že okno nášho programu sa po spustení objaví v strede obrazovky. Do vlastnosti Caption vpíšeme reťazec, k „Editor“ (samozrejme bez úvodzoviek). Potom pridáme komponent MainMenu, klikneme pravým tlačidlom a otvoríme Menu Designer. Uvidíme jednoduché prázdne menu (obr. 4). Do vlastnosti Caption vpíšeme reťazec „Súbor“.

Potom stlačíme klávesu Ins, čím do novovytvoreného menu „Súbor“ pridáme prázdnu položku. Presunieme svoju pozornosť k Object Inspectoru a do vlastnosti Caption nášho menu vpíšeme reťazec „Otvoriť“. Potom opäť stlačíme klávesu Ins a rovnakým spôsobom pridáme ďalšiu položku menu, tentoraz s textom „Uložiť ako...“. Zostáva už len dopísať položku „Skončiť“. Tá je však v každom slušnom programe oddelená od ostatných položiek čiarou. Ako túto čiaru vyrobiť? Veľmi jednoducho. Opäť stlačíme klávesu Ins a do vlastnosti Caption napíšeme znak - (t. j. mínus) a oddeľovacia čiara je na svete. Stačí už len pridať položku „Skončiť“ a menu je hotové.

Na formulár umiestnime komponent TPanel a nastavíme vlastnosť Align na alTop. Panel sa okamžite premiestni k hornej časti okna a bude sa dynamicky prispôsobovať jeho rozmerom. Nápis „Panel1“ vymažeme a na panel pridáme tri tlačidlá typu TBitBtn, ktorým vymažeme text z vlastnosti Caption, aby sme mali dosť miesta na obrázky. Potom priradíme každému tlačidlu ikonku (ja som použil obrázky nachádzajúce sa

Obr. 4

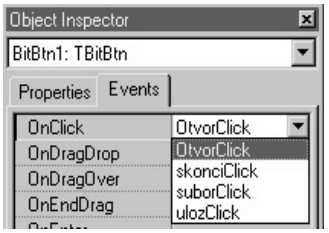
v adresári Delphi 3\Images\Buttons, konkrétne obrázky „Fileopen.bmp“, „Filesave.bmp“ a „Doorshut.bmp“).

Potom pridáme najdôležitejší komponent –TMemo. Vlastnosť Align nastavíme na alClient, čím sa komponent automaticky roztiahne na celú šírku okna. Z vlastnosti Lines potom vymažeme text „Memo1“.

Náš program bude používať štandardné okná na otváranie a ukladanie súborov, ktoré používa väčšina aplikácií Windows. Komponenty, ktoré nám použitie týchto okien umožnujú, sa nazývajú TOpenDialog a TSaveDialog. Pomocou vlastnosti Title priradíme každému dialógovému oknu nadpis (t. j. do vlastnosti Title komponentu TOpenDialog napíšeme „Otvoriť“ a do vlastnosti Title komponentu TSaveDialog napíšeme „Uložiť ako“).

Grafický návrh aplikácie je teda hotový, teraz ju musíme „rozhýbať“. Otvoríme Menu Designer a dvakrát klikneme na položku „Otvoriť“. Delphi vygeneruje nasledujúci kód:

```
procedure TForm1.Otvori1Click(Sender: TObject);
begin

end;
```

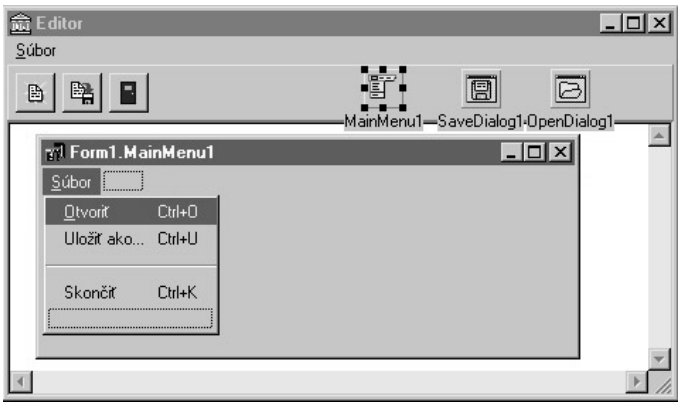


Obr. 5

Všimnite si, že Delphi odvodilo názov procedúry z názvu komponentu a vynechalo písmená s diakritikou. Ako som už spomínal, je dobrým zvykom prehľadne pomenúvať komponenty, pretože v opačnom prípade sa po pár mesiacoch nebudete vedieť vo svojich programoch orientovať. Preto sa ešte raz vrátime k Menu Designeru a všetkým trom položkám dáme prehladné názvy: *Otvor*, *Ulož*, *Skonči*. Názvy „OpenDialog1“ a „SaveDialog1“ sú dostatočne jasné, preto ich nebudeme meniť. Dvakrát klikneme na položku „Otvoriť“ a pridáme nasledujúci kód:

```
procedure TForm1.OtvorClick(Sender: TObject);
begin
     if OpenDialog1.Execute then
Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
end;
```

Funkcia Execute zabezpečí zobrazenie dialógového okna a vráti true, ak používateľ pri výbere súboru klikol na tlačidlo „Otvoriť“. Samotný súbor je načítaný pomocou metódy LoadFromFile vlastnosti Lines komponentu TMemo. Ak bolo dialógové okno otvorené, no používateľ sa z nejakého dôvodu rozhodol kliknúť na „Storno“, zvyšok kódu nevykoná (t. j. súbor sa nenačíta).



Obr. 6

Rutina na ukladanie súboru bude vyzerať podobne:

```
procedure TForm1.ulozClick(Sender: TObject);
begin
     if SaveDialog1.Execute then
Memo1.Lines.SaveToFile(SaveDialog1.FileName);
end;
```

Po otvorení ľubovolného súboru si všimneme jeden závažný nedostatok: ak sa chceme dostať na koniec dokumentu, musíme niekoľkokrát stlačiť kláves PgDn, čo je dosť nepohodlné. Nastavíme preto vlastnosť ScrollBars komponentu TMemo na ssBoth a program opäť spustíme. Po tejto víťanej zmene máme k dispozícii oba posuvníky (scroll bars) a v dokumente sa môžeme pohybovať oveľa komfortnejšie.

Naša aplikácia však ešte zďaleka nie je dokonalá. Nemá napríklad zmysel ukladať prázdny súbor, preto ak komponent TMemo neobsahuje nijaký text, mali by sme položku „Uložiť ako...“ v menu „Súbor“ zneprístupniť. Otvoríme teda Menu Designer, klikneme na položku „Súbor“ a do obslužnej rutiny udalosti OnClick pridáme tento kód:

```
procedure TForm1.suborClick(Sender: TObject);
begin
    if Memo1.Lines.Count=0 then uloz.Enabled:=false else
uloz.Enabled:=true;
end;
```

Pomocou vlastnosti Count zistíme, či obsahuje komponent TMemo nula riadkov (teda či je prázdny). Ak áno, potom bude položka „Uložiť ako...“ zneprístupnená.

Poslednou položkou v menu „Súbor“ je volba „Skončiť“. Obslužná rutina bude v tomto prípade veľmi jednoduchá:

```
procedure TForm1.skonciClick(Sender: TObject);
begin
     Application.Terminate;
end;
```

Pozorný čitateľ si teraz určite kladie otázku, načo sme do programu pridávali panel s tlačidlami. Väčšina používateľov Windows určite vie, že na vyvolanie jednej funkcie programu existuje viacero spôsobov. Napríklad v MS Word je možné otvoriť súbor pomocou volby „Otvoriť“ v menu „Súbor“, stlačením ikonky na nástrojovom pruhu alebo klávesovou skratkou Ctrl + O. Podobné vymoženosti teraz pridáme aj my do našej miniaplikácie. Predtým by som však chcel upozorniť začiatočníkov na jednu veľmi dôležitú vec: obslužné rutiny nie je potrebné písať dvakrát, treba len priradiť každému tlačidlu už existujúcu rutinu. Ako na to? Klikneme na príslušné tlačidlo, vyberieme udalosť OnClick a vyznačíme zodpovedajúcu rutinu (obr. 5). Našim tlačidlám teda postupne priradíme rutiny OtvorClick, UlozClick a SkonciClick.

A ako obohatiť program o klávesové skratky? Veľmi jednoducho. Otvoríme Menu Designer, vyberieme napr. položku „Otvoriť“ a pomocou vlastnosti Shortcut jej priradíme klávesovú skratku. Na tomto mieste by som sa ešte zmienil o špeciálnom použití znaku „&“ (ampersand) pri tvorbe menu. Ako všetci vieme, menu sú prístupné aj pomocou klávesu ALT. Napríklad v MS Word je možné otvoriť súbor kombináciou ALT + S (čím sa otvorí menu „Súbor“) a následným stlačením klávesu „O“. Naša miniaplikácia nesmie v nijakom prípade zaostávať za svetovými štandardmi, preto do nej túto vymoženosť okamžite zabudujeme. Otvoríme Menu Designer, klikneme na položku „Súbor“ a pred písmenko „S“ napíšeme znak „&“. Tento znak treba umiestniť vždy pred písmeno, ktoré nám má sprístupniť danú položku menu pomocou klávesnice. Umiestnime ho teda pred prvé písmenko položky „Otvoriť“. Po spustení nášho programu a stlačení klávesovej skratky ALT + S sa nám otvorí menu „Súbor“ a po následnom stlačení písmena „O“ sa otvorí dialógové okno „Otvoriť“.

Náš prvý miniprogram je teda hotový. Jeho finálnu podobu ukazuje obrázok č. 6. Má však ešte jednu závažnú logickú chybu. Na domácu úlohu skúste zistiť, o akú chybu ide.

Tretia časť: Databázy

V tejto časti sa začneme venovať problematike databáz, najskôr si však prezradíme riešenie domácej úlohy. Mali ste nájsť logickú chybu v našom miniprograme z predchádzajúcej časti seriálu. Riešenie je jednoduché a vlastne som ho načrtol už v predošlej časti: ak je komponent TMemo prázdny, položku „Uložiť ako...“ sme v hlavnom menu našej aplikácie znepristupnili, pretože ukladanie prázdneho súboru nemá zmysel. To isté platí aj pre tlačidlo, ktoré slúži na ukladanie súboru, a teda tiež nesmie byť prístupné, ak je komponent TMemo prázdny. V praxi to vyzerá takto: klikneme na druhé tlačidlo a premenujeme ho na SaveAs. Do obsluhy udalosti OnShow hlavného formulára pridáme nasledujúci kód:

```
procedure TForm1.FormShow(Sender: TObject);
begin
     SaveAs.Enabled:=false;
end
```

Tento kód nám zabezpečí, že po spustení programu bude tlačidlo SaveAs neprístupné. Len čo však napíšeme nejaký text, musíme ho opäť sprístupniť. Na tento účel použijeme udalosť OnChange komponentu TMemo, ktorá sa (ako to naznačuje jej názov) vyvolá vždy, keď sa v komponente vykoná nejaká zmena. Obsluha udalosti OnChange bude mať takúto formu:

```
procedure TForm1.Memo1Change(Sender: TObject);
begin
if Memo1.Lines.Count=0 then SaveAs.Enabled:=false
else SaveAs.Enabled:=true;
end;
```

Náš program spustíme a trochu ho preskúšame. V komponente TMemo sa nenachádza nijaký text, tlačidlo SaveAs je preto neprístupné. Z toho istého dôvodu je neprístupná aj položka „Uložiť ako...“ v menu „Súbor“. Tlačidlo SaveAs a položka „Uložiť ako...“ sa nám sprístupnia bezprostredne po napísaní prvého písmenka. Ak potom text z komponentu TMemo vymažeme, tlačidlo SaveAs a položka „Uložiť ako...“ budú opäť neprístupné, náš program teda funguje správne.

Databázy v Delphi

Delphi nám umožňuje jednoducho a flexibilne pristupovať k databázam najrôznejšieho „kalibra“, od lokálnych (Paradox, dBase) až po výkonné databázové servery (Interbase).

Skôr než sa začnem venovať tvorbe databáz, pokúsim sa vysvetliť, čo vlastne znamená pojem databáza. Databáza je používateľsky orientovaný súbor údajov, vnútorne štrukturovaný tak, aby umožňoval viacnásobný prístup k údajom. Každá databáza má svoju pevne stanovenú štruktúru. Základným prvkom štruktúry databázy sú polia. V závislosti od typu údajov, ktoré chceme v databáze používať, rozoznávame polia typu integer (celé čísla), float (desatinné čisla), alpha (textové reťazce) atď.

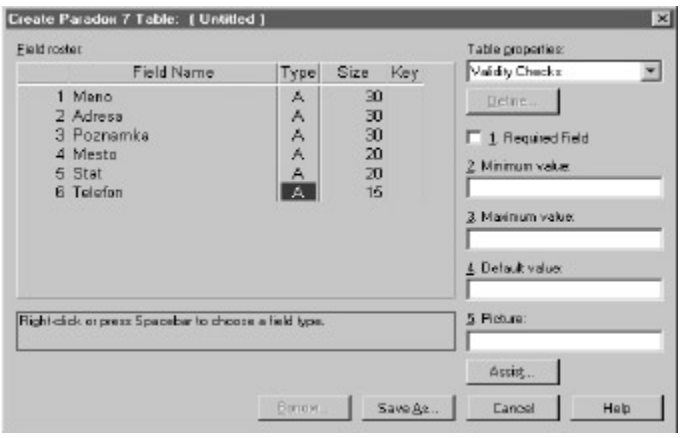
V súvislosti s databázami objasním dôležitý pojem alias. Alias je akýmsi skráteným pomenovaním databázy. Keby neexistovali aliasy, museli by sme napr. v dopytoch SQL používať názvy súborov spolu s cestou alebo by sme ich museli zdĺhavo vpisovať do vlastnosti DatabaseName komponentu TTable či TQuery. Dopyt SQL by potom vyzeral asi takto:

```
select * from c:\moje\pokusy\databazy\adresar order by meno.
```

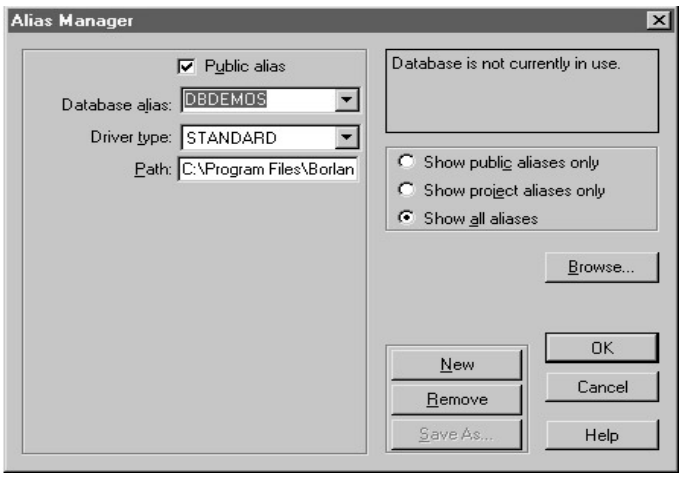
Predstavte si, že by ste museli napísať dopyt SQL, ktorý pracuje s piatimi súbormi!

Oveľa lepším riešením bude, ak vytvoríme alias s názvom adresar a priradíme mu názov súboru (samozrejme spolu s cestou). SQL dopyt bude potom omnoho prehľadnejší:

```
select * from adresar order by meno.
```



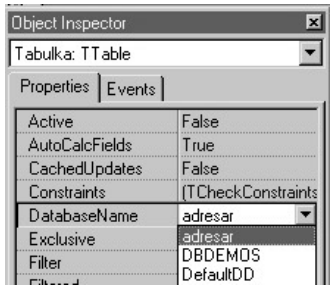
Obr. 1



Obr. 2

K dátam uloženým v tabuľke pristupujeme pomocou vlastností a metód komponentu TTable. Pokiaľ potrebujeme pracovať len so špecifickou množinou dát, ktorá spĺňa určité kritériá, použijeme komponent TQuery (pripomínam, že aj komponent TTable ponúka podobné možnosti, komponent TQuery je však v tomto smere flexibilnejší).

Základné databázové operácie budeme realizovať prostredníctvom komponentu TTable a niekoľkých vizuálnych komponentov, ktoré sú s ním úzko späté. Naším prvým databázovým programom bude jednoduchý adresár osôb. Databázu vytvoríme pomocou programu Database Desktop, ktorý je súčasťou Delphi.



Obr. 3

Spustíme Database Desktop a vytvoríme novú tabuľku (menu File - New - Table, v dialógovom okne Create Table vyberieme Paradox 7). Po vytvorení tabuľky postupne začneme pridávať polia. Prvé pole nazveme Meno, presunieme sa do stĺpčeka Type, stlačíme medzerník a vyberieme položku Alpha. Potom sa presunieme do kolónky Size, kde napíšeme číslo 30 (maximálna dĺžka tohto poľa teda bude 30 znakov). Kolónku Key zatiaľ necháme bez povšimnutia. Podobným spôsobom vytvoríme ďalšie polia: Adresa, Poznamka (obe s veľkosťou 30 znakov), Mesto, Stat (obe s veľkosťou 20 znakov) a pole Telefon s veľkosťou 15 znakov. Všetky polia budú typu Alpha (obr. 1). Tabuľku uložíme pod názvom Adresar. Po uložení vytvoríme alias, ktorý sa bude na našu tabuľku „odkazovať“: v Database Desktope z menu Tools vyberieme Alias Manager (obr. 2). Stlačíme New a do kolónky Database Alias napíšeme Adresar. Potom stlačíme tlačidlo Browse a vyberieme adresár, do ktorého sme našu databázu uložili a stlačíme OK. Database Desktop sa potom spýta, či má tento alias uložiť do súboru IDAPI32.CFG. Klikneme na Yes a alias je hotový. Teraz nám už zostáva iba napísať program, ktorý bude s databázou pracovať. Predtým si však povieme niečo o použití dátových modulov (data modules).

Dátové moduly

Keď vytvárame program v Delphi, všetky vizuálne i nevizuálne komponenty umiestňujeme na formulár. V prípade vizuálnych komponentov je to úplne v poriadku, pretože ich umiestnenie priamo súvisí s formulárom, na ktorom sa nachádzajú.

Horšie je to však s komponentmi typu TTable, TQuery alebo TDataSource. Predstavme si napríklad, že sme urobili jednoduchý program, ktorý umožňuje filtrovať databázu na základe určitého kritéria. Rutina, ktorá zabezpečuje filtrovanie, sa stane súčasťou formulára, na ktorom bude umiestnený komponent TTable alebo TQuery. Ak budeme chcieť náš kód použiť aj v iných programoch, neostane nám iná možnosť ako filtrovaciu rutinu pohľadať a nakopírovať ju do nového programu. Pritom existuje oveľa pohodlnejšie riešenie. Ak komponent TTable umiestnime na dátový modul, stane sa filtrovacia rutina súčasťou tohto datového modulu. Do nového programu potom stačí pridať odkaz na tento dátový modul (menu File – Use unit) a problém je vyriešený.

Náš prvý databázový program

Vytvoríme novú aplikáciu a na hlavný formulár umiestnime komponent TTable (pomenujeme ho Tabulka) a komponent TDataSource (pomenujeme ho dsTabulka). Vlastnosť DatabaseName komponentu TTable bude obsahovať náš novovytvorený alias Adresar (vyberieme ho zo zoznamu, obr. 3) a vlastnosť DataSet komponentu TDataSource nastavíme na Tabulka. Cestu k súboru teda komponent už pozná, potrebuje však „vedieť“ ešte názov súboru, s ktorým bude pracovať. Ten nastavíme pomocou vlastnosti TableName (vyberieme súbor Adresar.db) a vlastnosť Active nastavíme na true, čím tento súbor otvoríme. Teraz musíme pridať komponent, ktorý

nám umožní dáta zobraziť Týmto komponentom je TDBGrid (nachádza sa na palete Data Controls). Vlastnosť DataSource komponentu TDBGrid nastavíme na dsTabulka. Potom pridáme komponent TDBNavigator a vlastnosť DataSource nastavíme na dsTabulka. Tento komponent nám umožní vykonávať základné operácie s databázou: pohyb dopredu a dozadu, pridávanie, mazanie, potvrdzovanie zmien, rušenie zmien atď.

Náš prvý jednoduchý databázový program je prakticky hotový, môžeme ho teda spustiť a trochu sa s ním „pohrať“. Aby bol predsa len trochu vyspelejší, rozhodol som sa rozšíriť ho o možnosť filtrovania osôb v závislosti od štátu, v ktorom bývajú. Pridáme teda komponent TGroupBox a komponent TEdit, ktorý nazveme Stat. Ďalej pridáme zaškrtávacie políčko (komponent TCheckBox) a nazveme ho Aktivny.

Pri filtrovaní bude kľúčovú úlohu zohrávať obslužná rutina udalosti OnFilterRecord komponentu TTable. Táto sa aplikuje na všetky záznamy databázy a vyfiltruje len tie, ktoré spĺňajú požadované kritériá. Filtračná rutina bude vyzerať takto:

```
procedure TForm1.TabulkaFilterRecord(DataSet: TDataSet;
  var Accept: Boolean);
begin
accept:=AnsiLowerCase(Tabulka.FieldByName('Stat').AsString)=AnsiLowerCase(stat.text);
end;
```

Obr. 4

Pomocou funkcie AnsiLowerCase sa najprv reťazce prevedú na malé písmená. Ak by sme túto funkciu nepoužili, filter by fungoval, iba keby boli oba reťazce úplne zhodné. Vďaka konverzii je jedno, či napíšete slovensko, Slovensko alebo SlovenskO, program bude vždy pracovať správne. Po konverzii na malé písmená sa zisťuje, či je výraz

```
AnsiLowerCase(Tabulka.FieldByName
('Stat').AsString)=AnsiLowerCase
(stat.text)
```

pravdivý. Premenná accept bude potom nastavená v závislosti od pravdivosti tohto výrazu. Pokiaľ sa teda názov štátu získaný z práve filtrovaného záznamu zhoduje s názvom štátu zapísaným v komponente TEdit, výraz porovnávajúci tieto dve hodnoty nadobudne hodnotu true. A keďže medzi týmto výrazom a premennou je znamienko rovnosti, nadobudne táto premenná rovnakú hodnotu ako výraz, čiže hodnotu true. Obrázok č. 4 ukazuje náš program „v akcii“. Ako toto všetko súvisí sa dátovými modulmi? Všimnite si riadok procedure

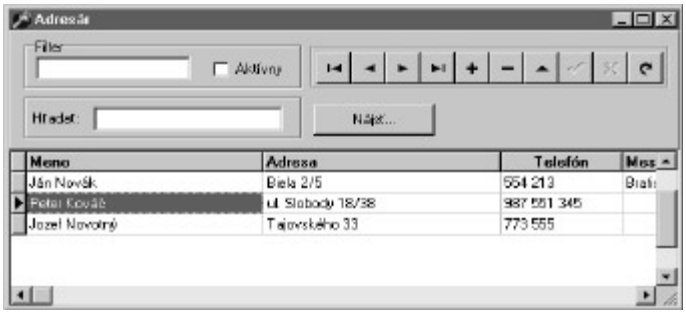
```
TForm1.TabulkaFilterRecord(DataSet: TDataSet; var Accept: Boolean);
```

Ako vidíme, obslužná rutina TForm1.TabulkaFilterRecord je súčasťou nášho formulára Form1. Ak by sme ju chceli znovu použiť v inom programe, museli by sme jej hlavičku (t. j.

```
TForm1.TabulkaFilterRecord(DataSet: TDataSet; var Accept: Boolean);)
```

vytvoriť ešte raz v novom programe a zo starého prekopírovať jej „telo“. Filtračných rutín však môže by oveľa viac, nehovoriac už o rutinách, ktoré môžu byť spojené s obsluhou iných udalostí. Ručné kopírovanie veľkého množstva kódu je nepohodlné a náchylné na chyby. Preto vývojári z Inprise vymysleli dátové moduly, ktoré pomáhajú riešiť práve takéto situácie. Pokiaľ databázové komponenty umiestnime na dátový modul, kód súvisiaci z databázami sa stane jeho súčasťou a dátový modul môžeme použiť v inej aplikácii. Upozorňujem, že na dátové moduly nie je možné umiestňovať vizuálne komponenty. Môžete ich však použiť aj na „skladovanie“ nevizuálnych komponentov, ktorými sú napr. TTimer alebo TImageList. Hoci sa nám už podarilo zvládnuť niektoré základné operácie s databázami, naša aplikácia má k dokonalosti veru ešte poriadne ďaleko. Určite ste si všimli, že napríklad nadpisy stĺpcov neobsahujú diakritiku (pretože sú to vlastne názvy polí), čo je veľmi závažný nedostatok. Ak si program maximalizujete na celú obrazovku, zistíte, že hoci sa hlavné okno zväčšilo, rozmery komponentu TDBGrid zostali nezmenené. Preto budeme náš program ďalej zdokonaľovať. V prvom rade pridáme do nášho projektu dátový modul, nazveme ho *dmAdresar* a premiestnime doň komponenty TTable a TDataSet. Ako vidíme, prepojenia s komponentom TDBGrid a TDBNavigator sa nám stratia, musíme teda opäť správne nastaviť vlastnosti všetkých komponentov.

Jediný rozdiel oproti predchádzajúcim nastaveniam bude, že vlastnosti DataSource komponentu TDBGrid a TDBNavigator budú obsahovať hodnoty *dmAdresar.dsTabulka*



Obr. 5

(dsTabulka nestačí, pretože komponent už nie je na formulári; dátových modulov môže byť v jednej aplikácii viac, preto musíme poskytnúť aj názov modulu, na ktorom sa komponent TDataSource nachádza). Filtračná rutina sa ‚bohužiaľ, na dátový modul nepremiestni spolu s komponentom TTable, preto jej telo budete musieť prekopírovať ručne a starú rutinu vymazať.

Nadpisy stĺpcov komponentu TDBGrid je možné meniť veľmi jednoducho: pravým tlačidlom klikneme na komponent TDBGrid a z lokálneho menu vyberieme Columns Editor. Stlačíme Add a Object Inspector nám zobrazí vlastnosti novovytvoreného objektu TColumn. Vlastnosť FieldName bude obsahovať názov poľa, ktorý sa má v príslušnom stĺpci zobrazovať. Okrem samotného nadpisu stĺpca môžete meniť font, ktorým bude tento nadpis zobrazený, šírku stĺpca, zarovnávanie atď. Ďalším problémom je neprispôsobenie sa veľkosti komponentu TDBGrid veľkosti hlavného formulára po jeho „natiahnutí“ či maximalizovaní. Riešenie podobného problému som načrtol už v predošlej časti seriálu, nebudem ho preto dopodrobna rozoberať.

Náš program zatiaľ umožňuje vykonávať iba tie najzákladnejšie databázové operácie, a teda nie je príliš užitočný. Preto ho teraz rozšírime o možnosť vyhľadávania podľa mena. Na hlavný formulár pridáme tlačidlo a pomenujeme ho najdi. Potom pridáme komponent TEdit, ktorý nazveme hladat. Na vyhľadávanie slúži metóda Locate, ktorá vracia true alebo false v závislosti od toho, či sa podarilo nájsť záznam spĺňajúci požadované kritériá. Obsluha udalosti OnClick tlačidla najdi bude vyzerať takto:

```
procedure TForm1.najdiClick(Sender: TObject);
var nasiel:boolean;
begin
nasiel:=dmAdresar.Tabulka.Locate('meno',hladat.Text,[loCaseInsensitive]);
  if not nasiel then Application.MessageBox('Hľadaný záznam sa
nepodarilo nájsť','Adresár',MB_OK or MB_ICONEXCLAMATION);
end;
```

Prvým parametrom metódy Locate je meno poľa, ktoré sa má prehľadávať. Ak potrebujeme prehľadávať viacero polí, ich názvy oddelíme bodkočiarkou. Ďalším parametrom je hodnota, ktorá sa má vyhľadať. Posledný a najzaujímavejší je parameter v odbornej dokumentácii nazývaný Locate options. Môže obsahovať hodnoty `loCaseInsensitive` alebo `loPartialKey`, prípadne obidve naraz. Ich význam je nasledujúci:

`loCaseInsensitive` – pri hľadaní sa neberie ohľad na malé a veľké písmená

`loPartialKey` – vyhľadá sa prvý záznam, ktorý obsahuje aspoň niekoľko začiatočných znakov parametru hodnota.

Ak z nášho príkladu vynecháme `loCaseInsensitive`, budeme musieť napísaťmeno hľadanej osoby do slova a do písmena tak, ako ho máme zapísané v adresári, v opačnom prípade nám ho program nenájde. Aj v prípade, ak nenapíšeme celé meno hľadanej osoby, vyhľadávanie bude neúspešné. Preto našu aplikáciu opäť trošku vylepšíme:

```
procedure TForm1.najdiClick(Sender: TObject);
var nasiel:boolean;
begin
nasiel:=dmAdresar.Tabulka.Locate('meno',hladat.Text,[loCaseInsensitive,loPartialKey]);
     if not nasiel then Application.MessageBox('Hľadaný záznam sa
nepodarilo nájsť','Adresár',MB_OK or MB_ICONEXCLAMATION);
end;
```

Po týchto úpravách sa už nemusíte starať o to, či píšete veľkými alebo malými písmenami. Dokonca ani nemusíte napísať meno hľadanej osoby v plnom znení, postačí niekoľko začiatočných znakov. Konečná verzia programu je na obr. 5.

Pre úplnosť ešte opíšem princíp práce s funkciu MessageBox triedy TApplication. Prvým parametrom je samotný text správy, ktorý sa má v dialógovom okne zobraziť, druhým parametrom je nadpis, ktorý toto okno bude mať. Tretí parameter sa nazýva Flags a obsahuje rôzne nastavenia okna, ako napr. ikonu, ktorú bude okno zobrazovať, ktoré tlačidlo má byť implicitné (ak ich je viac) atď. Podrobnejšie informácie nájdete v dokumentácii API funkcie MessageBox.

Naša prvá databázová aplikácia je teda na svete. Na domácu úlohu pouvažujte nad jej nedostatkami. V budúcej časti si niektoré z nich preberieme a postupne ich budeme odstraňovať.

Štvrtá časť: Metamorfózy

Možno vás prekvapil trochu netradičný názov tejto časti seriálu. Dôvodom pre takéto poetické pomenovanie je fakt, že naša miniaplikácia z predchádzajúcej časti bude podrobená radikálnym zmenám. Predpokladám, že ste po prečítaní predošlých častí už nadobudli určitú prax v práci s Delphi, preto počnúc týmto článkom už nebudem pracovný postup opisovať tak podrobne ako predtým. Na domácu úlohu ste mali porozmýšľať nad nedostatkami nášho programu. Teraz si niektoré z nich preberieme a postupne ich odstránime. Ešte prv sa však pokúsim objasniť problematiku indexov, ktorá bude na pochopenie niektorých pasáží môjho výkladu veľmi dôležitá.

Indexy

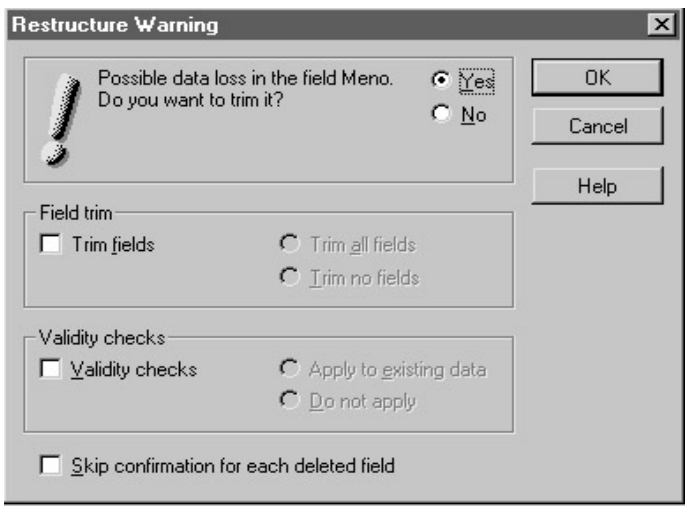
Počas našej „Delphi kariéry" sa často stretneme s problémom zoraďovania dát podľa určitých kritérií. Predstavme si napríklad, že pracujeme s tabuľkou, ktorá obsahuje geografické údaje o rôznych štátoch. Ak chceme zistiť, ktoré štáty majú najviac obyvateľov, potrebujeme údaje zoradiť podľa počtu obyvateľov. Ak si chceme nechať zobraziť štáty s najväčšou rozlohou, musíme ich mať zoradené podľa rozlohy. Údaje sú však do tabuľky vkladané takpovediac „náhodne", nie sú teda zoradené podľa nijakého kritéria. Aby sme náš problém vyriešili, potrebujeme sa oboznámiť s indexmi. Index je vlastne súbor, ktorý obsahuje vopred utriedené poradie jednotlivých záznamov bázydát. Komponent TTable si na zák- lade indexového súboru „zrekonštruuje" správne poradie záznamov. Indexy slúžia aj na urýchľovanie vyhľadávacích operácií, v niektorých prípadoch (napr. pri použití metódy SetRange) sú dokonca nevyhnutné (t. j. metóda funguje len pre indexované polia).

Ak chceme dáta zoradiť napríklad podľa rozlohy, musíme vytvoriť index na pole, ktoré reprezentuje rozlohu. Delphi (či presnejšie komponent TTable) však musí vedieť názov indexu, ktorý chceme použiť (keďže jedna bázadát môže mať viacero indexov). Názov indexu nastavujeme pomocou vlastnosti IndexName (to neplatí pre primárne indexy, ktoré nie je možné pomenovať). Ak pracujeme s tabuľkami typu dBase, musí byť index uvedený vlastnosťou IndexName súčasťou tzv. master index súboru, prípadne môže byť umiestnený v indexovom súbore určenom vlastnosťou IndexFiles.

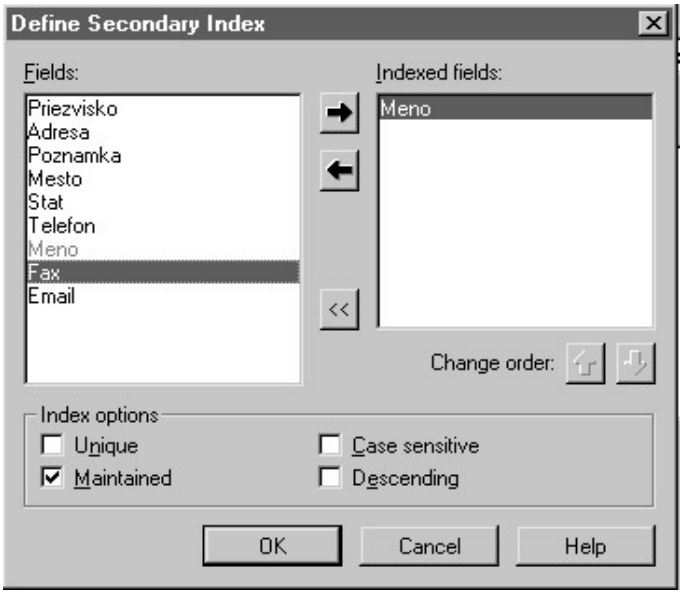
Databázový systém Paradox rozlišuje dva typy indexov: primárne a sekundárne. Primárny index sa nazýva aj kľúč (key). Primárny index je hlavný index, ktorý určuje poradie zobrazenia záznamov. V Database Desktope ho bezpečne spoznáte podľa hviezdičky, ktorá je umiestnená vedľa názvu indexovaného poľa (v kolónke Key). Tento index zároveň zabraňuje duplicite údajov. Ak sa pokúsite do takto indexovaného poľa zapísať hodnotu, ktorá v bázedát už existuje, vznikne výnimka Key violation. Pole, na ktoré sa bude primárny index vzťahovať, musí byť vždy prvé v poradí. Samozrejme, je možné mať primárny index aj na viacerých poliach, musia však nasledovať bezprostredne za prvým poľom. Alternatívnu možnosť tvoria sekundárne indexy, ktoré možno používať v kombinácii s primárnymi indexmi. Na rozdiel od primárnych indexov však musia mať sekundárne indexy názov.

Začíname vylepšovať

Prvým vylepšením nášho programu bude prehľadné triedenie mien podľa priezviska, čo si, samozrejme, vyžaduje použitie indexu. Skôr ako tento index vytvoríme, pridáme do bázydát ďalšie polia, ktoré by správny adresár osôb mal obsahovať. Aby nás staré údaje v bázedát zbytočne nerozptyľovali, tabuľku vyprázdnime (v Database Desktope zvolíme Tools – Utilities – Empty). Pokiaľ však máme súčasne spustené Delphi, nesmieme zabudnúť tabuľku zavrieť (t. j. vlastnosť Active nastavíme na false). Po vyprázdnení tabuľku otvoríme a z menu Table vyberieme Restructure. Potom pridáme nasledujúce polia: fax (s dĺžkou 15 znakov) a email (s dĺžkou 30 znakov). Prvé pole (teda pole meno) premenujeme na priezvisko a veľkosť nastavíme na 30 znakov. Presunieme sa na kolónku Key a stlačíme hviezdičku (*). Nakoniec pridáme pole meno a veľkosť nastavíme na 15 znakov.

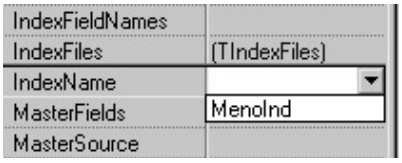


Obr. 1



Obr. 2

Po podobných úpravách štruktúry bázydát sa vám môže stať, že sa zobrazí varovanie podobné ako na obrázku č. 1. Ak ste totiž zmenili pôvodnú dĺžku poľa na kratšiu, môže dôjsť k strate dát. Naša tabuľka je však prázdna, o nijaké údaje teda neprídeme, preto klikneme na OK. Potom opäť vyberieme Restructure z menu Table a z rozbaľovacieho zoznamu Table properties vyberieme Secondary indexes a klikneme na tlačidlo Define. Zobrazí sa nám zoznam polí našej tabuľky. Vyberieme pole meno (pretože osoby budeme zoraďovať voliteľne buď podľa krstného mena, alebo podľa priezviska) a klikneme na šípku smerujúcu na pravú stranu (obr. 2). Po stlačení tlačidla OK nás Database Desktop požiada, aby sme novovytvorený index pomenovali. Nazveme ho teda MenoInd a klikneme na OK. Ak máte chuť trocha experimentovať, stlačením klávesu F9 môžete pridávať záznamy priamo v Database Desktope. Tabuľku potom uložíme a zatvoríme.



Obr. 3

Presunieme sa späť do Delphi, kde tabuľku otvoríme, spustíme program a pridáme niekoľko mien. Vďaka indexom sa nám z neprehľadnej kôpky dát podarilo „vyčarovať" prehľadný zoznam zoradený podľa abecedy. Pozorní čitatelia akiste nezabudli, že index sme vytvorili aj na pole meno. Preto program ukončíme a v komponente TTable nastavíme sekundárny index MenoInd (obr. 3), ktorý nám umožní zoradiť zoznam osôb podľa krstných mien. Po spustení programu vidíme, že záznamy sú skutočne sortované podľa krstných mien. Náš program je síce funkčný, ale stále má podobu akéhosi „laboratórneho experimentu". Preto teraz radikálne zmeníme jeho vzhľad.

V prvom rade odstránime všetky prvky súvisiace s vyhľadávaním a filtrovaním (vrátane obslužných rutín). Odstránime aj komponent TDBNavigator a ponecháme iba komponent TDBGrid a horný panel. Pomocou Columns Editora odstránime z komponentu TDBGrid všetky stĺpce okrem priezviska, mena a adresy. Svoju pozornosť potom presunieme k Object Inspectoru, konkrétne k vlastnosti Options komponentu TDBGrid. Ako vidíme, je tu množstvo podvlastností, ktoré budeme v budúcnosti často používať, preto si ich teraz postupne preberieme.

Vlastnosť dgEditing určuje, či komponent povolí editovanie dát. Nastavíme ju na false, pretože budeme údaje editovať pomocou iných komponentov a na inom formulári. Nastavenie dbAlwaysShowEditor na true spôsobí, že komponent sa vždy bude nachádzať v editovacom režime, t. j. nemusíme stlačiť Enter či F2, aby sme komponent do tohto režimu prepli.

Vlastnosť dgTitles určuje, či sa nad jednotlivými stĺpcami budú zobrazovať nadpisy. Vlastnosť dgIndicator zapína a vypína malý trojuholníček, ktorý nám označuje aktuálny záznam. Nastavíme ju na false. Vlastnosť dgColumnResize povoľuje/zakazuje zmenu veľkosti stĺpcov a presúvanie stĺpcov počas behu programu. Nastavíme ju na true. Vlastnosť dgColLines povoľuje/zakazuje vykresľovanie oddeľovacích čiar medzi stĺpcami. Vlastnosť dgRowLines povoľuje/zakazuje vykreslenie oddeľovacej čiary medzi riadkami. Nastavenie vlastnosti dgTabs sprístupní navigáciu pomocou klávesov Tab a Shift + Tab. Vlastnosť dgRowSelect umožňuje vyznačiť celý riadok. Nastavíme ju na true. Vlastnosť dgAlwaysShowSelection povoľuje/zakazuje zobrazenie tzv. focus rectangle, keď komponent práve nemá fokus. Túto vlastnosť nastavíme na true.

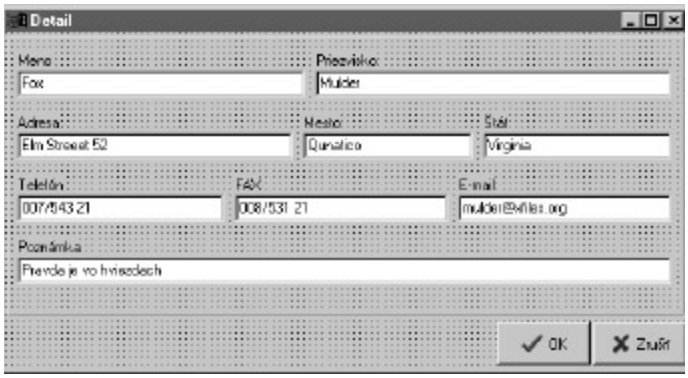
Pokročilejší mi teraz, dúfam, odpustia malé odbočenie. Pokúsim sa začiatočníkom vysvetliť, čo to vlastne ten fokus je. Spustíme Notepad (po slovensky Poznámkový

blok) a zvolíme Otvoriť. Presunieme sa do rozbaľovacieho zoznamu Súbory typu (Files of type) a zvolíme formát TXT. Akiste ste si všimli, že po tomto úkone sa farba pozadia vnútri rozbaľovacieho zoznamu zmenila. Hovoríme, že rozbaľovací zoznam má fokus. Je potrebné povedať, že fokus môže mať vždy len jeden prvok na obrazovke. Vyberieme ľubovoľný súbor, stlačíme tlačidlo Otvoriť a podržíme ho. V momente stlačenia nadobudlo pozadie rozbaľovacieho zoznamu opäť pôvodnú farbu. Fokus sa teraz presunul na tlačidlo Otvoriť. Na rozdiel od zoznamu tlačidlo nezmení farbu, iba sa pri jeho okrajoch vykreslí obdĺžnik z tenkej prerušovanej čiary, tzv. focus rectangle. Myšou sa presunieme čo najďalej od tlačidla Otvoriť a pustíme ho. Potom stlačíme a podržíme tlačidlo Zrušiť. V momente stlačenia zmizne focus rectangle z tlačidla Otvoriť a presunie sa na tlačidlo Zrušiť Vráťme sa však späť do Delphi.

Nastavenie vlastnosti dgConfirmDelete určuje, či sa po stlačení kombinácie Ctrl + Delete má zobraziť dialógové okno požadujúce potvrdenie zmazania (t. j. „Delete record?“ OK/Cancel). Pomocou vlastnosti dgCancelOnExit určíme, či po ukončení práce z TDBGridom (t. j. keď sa fokus presunie na iný komponent) majú byť z bázydát odstránené prázdne záznamy. Vlastnosť dgMultiSelect dovoľuje/zakazuje vyznačenie viacerých riadkov (robí sa to pomocou klávesu Ctrl). Stručný prehľad najdôležitejších vlastností Options máme teda za sebou.

Doteraz sme údaje zobrazovali, pridávali a editovali pomocou komponentov TDBGrid a TDBNavigator. Musím však podotknúť, že komponenty za vás program nenapíšu. Nemôžete sa stále spoliehať na to, že vytvoríte aplikáciu bez jediného riadka kódu. V niektorých prípadoch to síce možné je, výsledok však podľa mňa pôsobí dosť amatérskym dojmom. To je hlavný dôvod, prečo som sa rozhodol zmeniť dizajn nášho adresára osôb.

Pustíme sa teda do práce. Vytvoríme nový formulár a nazveme ho detail. Do vlastnosti Caption napíšeme „Detail“ (bez úvodzoviek), vlastnosť BorderStyle nastavíme na bsDialog a vlastnosť Position nastavíme na poScreenCenter. Formulár uložíme pod názvom fdetail. Potom pridáme dve tlačidlá TBitBtn. Prvé nazveme dobre, vlastnosť Kind nastavíme na



Obr. 4

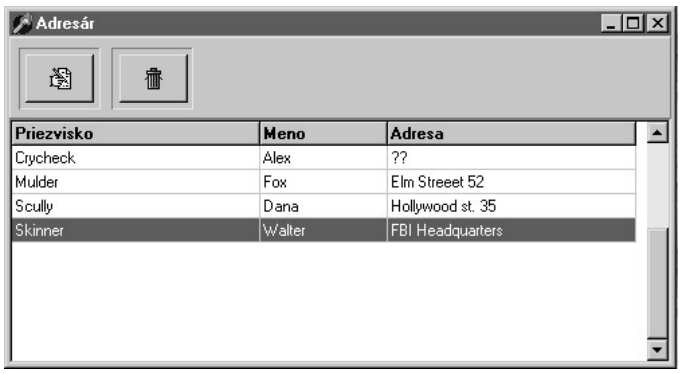
bkOk. Po tomto úkone sa v tlačidle automaticky objaví nápis „OK“ a vlastnosť ModalResult sa nastaví na mrOk. Druhé tlačidlo nazveme zrus a vlastnosť Kind nastavíme na bkCancel. Podobne ako v predchádzajúcom prípade aj teraz dôjde k automatickému nastaveniu vlastnosti Caption a vlastnosti ModalResult. Nám sa však anglofónne znejúci nápis „Cancel“ nepozdáva, preto ho zmeníme na rýdzo slovenský – „Zrušiť“. Aby dialógové okno vyzeralo trošku krajšie, „orámujeme“ ho komponentom TBevel.

Základný návrh formulára je teda hotový, zostáva nám ešte pridať komponenty, ktoré nám umožnia prezerať a editovať údaje z bázydát. Na formulár preto pridáme komponent TDBEdit. Podobe ako komponent TDBGrid aj komponent TDBEdit potrebuje vedieť, z ktorého dátového zdroja (data source) má čerpať údaje. Na rozdiel od TDBGridu však vie zobraziť iba jedno pole, ktoré nastavíme pomocou vlastnosti DataField. Nastavíme teda vlastnosť DataSource na *dmAdresar.dsTabulka* a vyberieme pole meno. Ak sme správne pracovali, komponent nám zobrazí krstné meno prvej osoby zapísanej v bázedát. Podobným spôsobom pridáme ďalšie komponenty TDBEdit a každému priradíme jedno pole bázydát. Samozrejmosťou sú komponenty TLabel, ktoré nám sprehľadnia orientáciu vo formulári. Konečný výsledok vidíme na obrázku č. 4.

Návrh formulára je síce ukončený, musíme však ešte pridať kód, ktorý nám ho po dvojitom kliknutí na komponent TDBGrid zobrazí:

```
procedure TForm1.DBGrid1DblClick(Sender: TObject);
begin
     Detail.ShowModal;
end;
```

Pri pokuse o spustenie programu nám Delphi oznámi, že sme použili formulár detail nachádzajúci sa v unite fdetail, a súčasne sa nás pýta, či si prajeme tento unit pridať do uses klauzuly v unite formulára Form1. (t. j. zobrazí sa dialógové okno s textom Form ‚Form1‘ references form ‚detail‘ declared in unit ‚fdetail‘ which is not in your USES list. Do you wish to add it?). Klikneme OK a program spustíme. Dáta umiestnené



Obr. 5

v komponentoch TDBEdit síce možno editovať, vykonané zmeny sa však nedajú zrušiť. Inými slovami, nezáleží na tom, či kliknete na tlačidlo OK alebo Zrušiť, zmeny sa tak či tak zapíšu. No tento „neduh“ veľmi rýchlo odstránime.

Predtým si však treba objasniť, na čo slúži vlastnosť ModalResult komponentov TButton, TBitBtn a TForm. V prípade komponentov TButton a TBitBtn ModalResult stanovuje, na akú hodnotu bude nastavená vlastnosť ModalResult formulára, na ktorom sú tieto komponenty umiestnené. Hodnoty vlastnosti ModalResult môžu byť nasledujúce: mrNone, mrOk, mrCancel, mrAbort, mrRetry, mrIgnore, mrYes, mrNo a mrAll. Na čo je to dobré? Akiste sa pamätáte, že náš program ukladá zmeny do bázydát bez ohľadu na to, či sme stlačili OK alebo Zrušiť. Vďaka vlastnosti ModalResult môžeme zistiť, ktoré z týchto tlačidiel bolo stlačené, a podľa toho sa budeme riadiť. Okrem toho nastavenie vlastnosti ModalResult na hodnotu inú ako mrNone má za následok zatvorenie formulára, takže odpadá nutnosť volania metódy Close. Na zapísanie vykonaných zmien do bázydát slúži metóda Post, na zrušenie zmien slúži metóda Cancel (obe patria komponentu TTable). Náš program teraz upravíme tak, že po stlačení tlačidla OK zapíše zmeny do bázydát a po stlačení tlačidla Zrušiť všetky zmeny odvolá:

```
procedure TForm1.DBGrid1DblClick(Sender: TObject);
begin
     Detail.ShowModal;
     if detail.ModalResult=mrOk then dmAdresar.tabulka.Post;
     if detail.ModalResult=mrCancel then dmAdresar.tabulka.Cancel;
end;
```

Po spustení sa aplikácia síce bude správať korektne, pokiaľ však formulár iba zobrazíme a nevykonáme nijaké zmeny údajov, po stlačení OK alebo Zrušiť vznikne výnimka s chybovým hlásením Dataset not in edit or insert mode. Aby sme pochopili, kde sa stala chyba, musíme sa oboznámiť s vlastnosťou State komponentu TTable.

Pri práci s údajmi komponent TTable pracuje v rôznych stavoch, ktoré závisia od toho, čo práve robíme. Kým je bázadát zatvorená (neaktívna), vlastnosť State má hodnotu dsInactive. Po otvorení súboru si údaje najprv iba prezeráme, tabuľka je teda v prezeracom režime; vlastnosť State má hodnotu dsBrowse. Pokiaľ sme sa rozhodli dáta editovať, tabuľka sa prepne do editovacieho režimu a vlastnosť State nadobudne hodnotu dsEdit. Pri pridávaní údajov bude tabuľka prepnutá do režimu pridávania a vlastnosť State nadobudne hodnotu dsInsert.

Okrem týchto stavov má tabuľka aj iné stavy (nap. dsSetKey, dsCalcFields), o tých si však povieme inokedy. Je zrejmé, že nemá zmysel volať metódu Post v iných stavoch ako dsEdit a dsInsert, pretože k zmene údajov dochádza jedine v týchto dvoch stavoch. Preto musíme pred volaním metódy Post zistiť, či sa tabuľka nachádza v niektorom z týchto stavov:

```
procedure TForm1.DBGrid1DblClick(Sender: TObject);
begin
     Detail.ShowModal;
     if (dmAdresar.tabulka.State=dsEdit)
     or (dmAdresar.tabulka.State=dsInsert)then
     begin
          if detail.ModalResult=mrOk then dmAdresar.tabulka.Post;
          if detail.ModalResult=mrCancel then
dmAdresar.tabulka.Cancel;
     end;
end;
```

Všimnite si, že program funguje správne, aj keď miesto tlačidla Zrušiť stlačíte tlačidlo x v pravom hornom rohu okna (t. j. tlačidlo určené na zatváranie okien vo všetkých aplikáciách Winows). Naša aplikácia síce vyzerá už trošku životaschopnejšie, stále jej však chýba možnosť pridávania a mazania údajov. Kým sme používali TDBNavigator, bol to práve on, kto robil „špinavú robotu“ za nás. No teraz sa budeme musieť zaobísť aj bez neho. Pridáme teda na hlavný formulár dve tlačidlá typu TBitBtn, prvé nazveme pridaj a druhé zmaz. Je na vás, aké ikonky im priradíte.

Nepochybujem, že váš zmysel pre estetický návrh aplikácie je oveľa lepšie vyvinutý ako môj, preto nechávam na vás aj umiestnenie tlačidiel. Na prepnutie tabuľky do stavu dsInsert slúži metóda Append; po jej zavolaní bude bázadát pripravená prijať ďalší záznam. Obslužná rutina udalosti OnClick tlačidla pridaj bude vyzerať takto:

```
procedure TForm1.PridajClick(Sender: TObject);
begin
     dmAdresar.tabulka.Append;
     Detail.ShowModal;
     if Detail.ModalResult=mrOk then dmAdresar.tabulka.Post;
     if Detail.ModalResult=mrCancel then dmAdresar.tabulka.Cancel;
end;
```

Všimnite si, že tentoraz netestujeme, či je bázadát v dsEdit alebo dsInsert režime. Nepotrebujeme to zistiť z toho dôvodu, že sme ju do „pridávacieho“ režimu sami prepli. V súvislosti s pridávaním spomeniem ešte jednu veľmi dôležitú vec: pole priezvisko je v bázedát definované ako kľúčové (key field). To znamená, že pri pokuse o dvojnásobné zapísanie toho istého priezviska (presnejšie povedané, pri volaní metódy Post) vznikne výnimka Key violation. Nám sa však anglofónne chybové hlásenia nepáčia, preto vytvoríme takú obsluhu, ktorá vypíše slovenský preklad:

```
procedure TForm1.PridajClick(Sender: TObject);
begin
  try
     dmAdresar.tabulka.Append;
     detail.ShowModal;
     if detail.ModalResult=mrOk then dmAdresar.tabulka.Post;
     if detail.ModalResult=mrCancel then dmAdresar.tabulka.Cancel;
  except
     on E:Exception do
     if pos('key violation',AnsiLowerCase(e.message))>0 then
      begin
 dmAdresar.tabulka.Cancel;
 Application.MessageBox('Duplicitné zaradenie!','Chyba',
             MB_OK OR MB_ICONERROR);
         end
      else
       begin
        raise
       end;
end;
end;
```

Pamätáte sa ešte na prvú časť seriálu? Okrem iného som v nej spomenul, na čo sú slúži implicitná obsluha výnimky (default exception handler). Pre istotu to však zopakujem ešte raz. Pokiaľ v našom programe naozaj nastane výnimka Key violation, bude korektne obslúžená (t. j. metódou Cancel sa zrušia všetky zmeny a zobrazí sa slovenské chybové hlásenie). Ak však náhodou nastane výnimka iného druhu, „velenie“ preberá implicitná obsluha výnimky, ktorá zobrazí anglické chybové hlásenie. Implicitnú obsluhu vyvolá príkaz raise.

Pridávanie záznamov máme teda vyriešené, zostáva nám ešte implementovať vymazávanie, ktoré sa vykoná prostredníctvom metódy Delete:

```
procedure TForm1.zmazClick(Sender: TObject);
var odpoved:integer;
begin
     odpoved:=Application.MessageBox('Naozaj chcete vymazat  tento
záznam?','Mazanie',
               MB_YESNO OR MB_ICONQUESTION OR MB_DEFBUTTON2);
     if odpoved=idYes then dmAdresar.tabulka.Delete;
end;
```

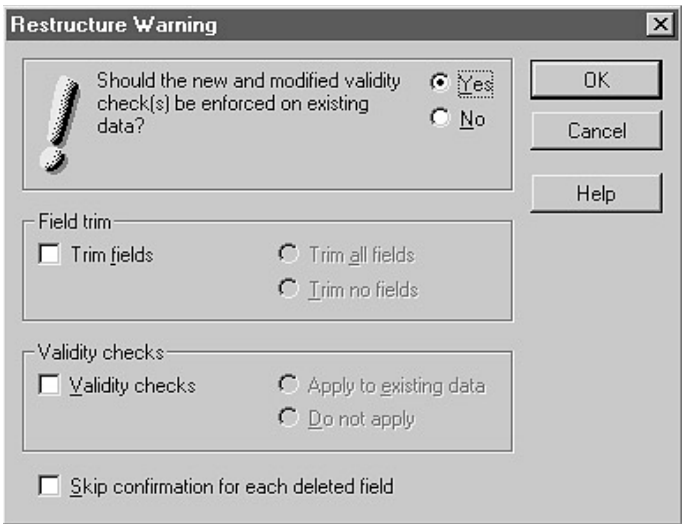
Možno si teraz položíte otázku, čo znamená to záhadné MB_DEFBUTTON2. Pomocou tejto konštanty zabezpečíme, že implicitne bude vyznačené druhé tlačidlo, teda tlačidlo Nie. Ak po zobrazení varovania stlačíme Enter, nastane ten istý efekt, ako keby sme stlačili tlačidlo Nie. Pokiaľ by totiž bolo implicitne vyznačené tlačidlo Áno, mohlo by sa stať, že sa po zobrazení varovania „uklepneme“ a záznam nedopatrením vymažeme. Konečný výsledok môjho esteticko-programátorského snaženia vidíte na obrázku č. 5.

Nadnes je to všetko. Náš program už tradične obsahuje dosť vážnu logickú chybu. Na domácu úlohu sa pokúste zistiť, o akú chybu ide. Prezradím iba toľko, že táto chyba súvisí s poľami meno a priezvisko.

Kam by ste sa mali pozrieť:
http://sunsite.icm.edu.pl/delphi – Delphi super page
http://www.torry.ru – Torry's Delphi pages
http://www.dataweb.net/~r.p.sterkenburg/indexpag.htm – Delphi bug list
http://delphree.clexpert.com – Iniciatíva pre rozvoj Open Source software v Delphi

Piata časť: Metamorfózy II

V tejto časti budeme pokračovať vo vylepšovaní. Na domácu úlohu ste mali nájsť pomerne vážnu logickú chybu v našej aplikácii. Na konci článku som vám našepkal, že táto chyba súvisí s poľami *meno* a *priezvisko*. Aby som nemárnil čas neúčelným chodením okolo horúcej kaše, prezradím, kde je pes zakopaný. Na svete existuje mnoho ľudí s rovnakým priezviskom. U nás sú veľmi rozšírené napr. priezviská Tóth a Kováč, v Amerike napríklad Smith a Anderson. Problém teda nastane, ak by sme si do adresára chceli napísať mená Gillian Anderson a Pamela Anderson. Keďže sme ako kľúčové pole nadefinovali pole *priezvisko*, program nám dovolí zapísať buď jednu, alebo druhú herečku, nie však obidve naraz.



Obr. 1

Preto musíme zmeniť štruktúru databázy. Spustíme Database Desktop a dáme sa do práce. Ako som už minule spomínal, kľúčové polia musia nasledovať bezprostredne za sebou, preto musíme pole *adresa* premenovať na *meno*, pole poznamka premenovať na *adresa* a druhé pole *meno* premenovať na *poznamka*. Do kolónky Key vedľa poľa *meno* umiestnime hviezdičku, čím ho „povýšime“ na kľúčové pole. Poradie a dĺžka polí bude teda nasledujúca: *priezvisko* (30 znakov), *meno* (15 znakov), *adresa* (30 znakov), *mesto* (20 znakov), *stat* (20 znakov), *telefon* (15 znakov), *poznamka* (30 znakov), *fax* (15 znakov) a *email* (30 znakov). Po týchto úpravách bude obsah databázy trochu pomiešaný: v poli *meno* sa bude zobrazovať adresa, pole adresa bude obsahovať poznámku a pole poznamka bude obsahovať krstné mená osôb. Tabuľku preto vyprázdnime a údaje zapíšeme znovu. V súvislosti s vyprázdňovaním spomeniem ďalší „neduh“ nášho programu: ak máme v databáze napríklad päť záznamov, musíme päťkrát stlačiť tlačidlo zmaz a päťkrát „odkliknúť“ varovanie. Existuje však oveľa pohodlnejší spôsob: metóda EmptyTable komponentu TTable. Pridáme teda na hlavný formulár nové tlačidlo a nazveme ho *vyprazdni*. Obsluha udalosti OnClick bude vyzerať takto:

```
procedure TForm1.vyprazdniClick(Sender: TObject);
var odpoved:integer;
begin
odpoved:=Application.MessageBox('Naozaj chcete databázu vyprázdnit?',
      'Varovanie',MB_YESNO OR MB_ICONQUESTION OR MB_DEFBUTTON2);
      if odpoved=idNo then exit;
      dmAdresar.tabulka.close;
dmAdresar.tabulka.Exclusive:=true;
dmAdresar.tabulka.EmptyTable;
dmAdresar.tabulka.Exclusive:=false;
      dmAdresar.tabulka.Open;
end;
```

V súvislosti s metódou EmptyTable vás musím upozorniť na jednu nepríjemnú skutočnosť: táto metóda nefunguje správne, pokiaľ ste vlastnosť Active v Object Inspectore nastavili na true. Po stlačení tlačidla dostanete chybové hlásenie Table is busy. Ak však Delphi ukončíte a náš program spustíte pomocou Explorera či Windows Commandera, operácia prebehne korektne. Alternatívnou možnosťou je otvorenie tabuľky počas behu programu metódou Open. Žiaľ, ani Delphi nie je bez chýb – a toto je jedna z tých menej vážnych. Ak sa chcete s „muchami“ Delphi oboznámiť podrobnejšie, pozrite si WWW stránku Delphi Bug List, nájdete ju na adrese http://www.dataweb.net/~r.p.sterkenburg/indexpag.htm.

Zamyslime sa teraz na chvíľu nad našou „vyprázdňovacou“ rutinou. Doteraz sme pristupovali k vlastnostiam a metódam komponentov pomocou zdĺhavých konštrukcií typu dmAdresar.Tabulka.close. Našťastie, vývojárom z firmy Inprise nie je naše pohodlie ľahostajné, a tak nám písanie kódu zjednodušili príkazom with. Nová verzia našej „vyprázdňovacej“ rutiny bude vyzerať takto:

```
procedure TForm1.vyprazdniClick(Sender:
TObject);
var odpoved:integer;
begin
     odpoved:=Application.MessageBox('Naozaj
chcete databázu vyprázdnit?',
      'Varovanie',MB_YESNO OR MB_ICONQUESTION
OR MB_DEFBUTTON2);
     if odpoved=idNo then exit;
     with dmAdresar.tabulka do
     begin
          Close;
          Exclusive:=true;
          EmptyTable;
          Exclusive:=false;
          Open;
     end; //with
end;
```

Všimnite predposledný riadok. Aby som uľahčil orientáciu v programe, urobil som si poznámku, že príkaz end sa vzťahuje na blok začínajúci sa príkazom with. Odporúčam, aby ste si zvykli za príkaz end písať poznámku, na čo sa tento príkaz vzťahuje. V opačnom prípade sa totiž môže stať, že pri zložitých konštrukciách nebudete vedieť, či príkaz end ukončuje blok try...except, či je súčasťou bloku try...finally atď. Mnohí z vás si možno položili otázku, čo sa stane, ak sa pokúsime vymazať záznam z prázdnej databázy, alebo ak sa pokúsime prázdnu databázu vyprázdniť. V druhom prípade sa nestane nič, v prvom prípade vznikne výnimka s chybovým hlásením Cannot perform this operation on an empty dataset. Samozrejme, niet sa čomu čudovať, veď z prázdnej databázy nie je čo vymazať. Riešenie tohto problému je jednoduché: ak je databáza prázdna, tlačidla zmaz a vyprazdni jednoducho znepristupníme. Na tento účel som vytvoril procedúru UpdateButtons, ktorej kód vyzerá takto:

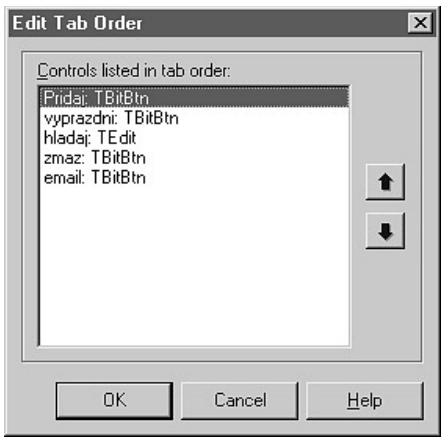
```
procedure TForm1.UpdateButtons;
begin
     if dmAdresar.tabulka.RecordCount=0 then
     begin
        zmaz.Enabled:=false;
        vyprazdni.Enabled:=false;
     end
     else
     begin
        zmaz.Enabled:=true;
        vyprazdni.Enabled:=true;
     end;// if RecordCount=0
end;
```

Aby bol program funkčný, musíme túto procedúru zavolať bezprostredne po vymazaní záznamu. V podstate existujú dve riešenia tohto problému. Prvé spočíva v zavolaní procedúry UpdateButtons hneď po vykonaní metódy Delete:

```
procedure TForm1.zmazClick(Sender: TObject);
var odpoved:integer;
begin
   try
      odpoved:=Application.MessageBox('Naozaj
chcete vymazat tento záznam?','Mazanie',
                MB_YESNO OR MB_ICONQUESTION
OR MB_DEFBUTTON2);
                  if   odpoved=idYes   then
dmAdresar.tabulka.Delete;
      UpdateButtons;
      except
         raise
      end;// except
end;
```

Druhé riešenie je založené na umiestnení volania tejto procedúry do obsluhy udalosti OnDelete komponentu TTable:

```
procedure
TdmAdresar.tabulkaAfterDelete(DataSet:
TDataSet);
begin
     Form1.UpdateButtons;
end;
```



Obr. 2

Ktorému riešeniu teda dáme prednosť? Jednoznačne druhému, pretože je podstatne prehľadnejšie. Pokiaľ použijeme riešenie číslo jeden vo väčšej aplikácii a budeme chcieť zistiť, čo všetko program vykoná po vymazaní záznamu, budeme musieť vyhľadať všetky výskyty metódy Delete v celom projekte a pozrieť sa na kód, ktorý za nimi nasleduje. Ak však použijeme variant číslo dva, po jedinom dvojkliknutí máme prehľad o kóde, ktorý sa vykoná po zmazaní záznamu. Aby náš program pracoval správne, nesmieme po vyprázdnení databázy zabudnúť zavolať procedúru UpdateButtnos:

```
procedure TForm1.vyprazdniClick(Sender:
TObject);
var odpoved:integer;
begin
   try
      odpoved:=Application.MessageBox('Naozaj
chcete databázu vyprázdnit?',
       'Varovanie',MB_YESNO OR MB_ICONQUESTION
OR MB_DEFBUTTON2);
      if odpoved=idNo then exit;
     with dmAdresar.tabulka do
     begin
          Close;
          Exclusive:=true;
          EmptyTable;
          Exclusive:=false;
          Open;
     end;//with
    UpdateButtons;
   except
      raise;
   end;//except
end;
```

Bystrejší a skúsenejší používatelia Delphi si určite všimli, že volanie procedúry UpdateButtnos v obsluhe udalosti OnDelete je síce na správnom mieste, je však potrebné umiestniť ho aj inde. Urobme nasledujúci pokus: vyprázdnime databázu. Tlačidlá zmaz a vyprazdni sa stanú neprístupnými. Potom pridajme jeden záznam a skúsme ho vymazať. Samozrejme, nepodarí sa nám to, pretože obe „inkriminované“ tlačidlá ostali neprístupné. Tento nedostatok odstránime volaním procedúry UpdateButtnos v obslužnej rutine udalosti AfterPost, ktorá sa vyvolá vždy po volaní metódy Post. Obsluha udalosti AfterPost teda bude vyzerať takto:

```
procedure TdmAdresar.tabulkaAfterPost(DataSet:
TDataSet);
begin
          Form1.UpdateButtons;
end;
```

Ani napriek týmto úpravám však nie je náš program dostatočne „inteligentný“. Opäť vykonáme jeden malý experiment: databázu vyprázdnime, program ukončíme a spustíme znovu. Tlačítka zmaz a vyprazdni sú napriek nulovému obsahu databázy stále aktívne. Riešenie je veľmi jednoduché: procedúru UpdateButtnos musíme volať aj pri štarte programu. Na tento účel som použil udalosť OnShow formulára Form1, ktorej obslužná rutina bude vyzerať takto:

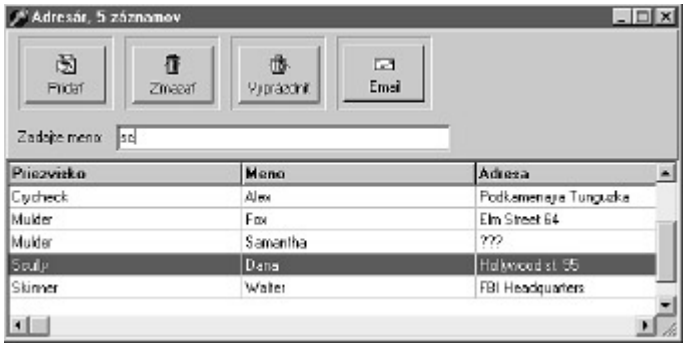
```
procedure TForm1.FormShow(Sender: TObject);
begin
     UpdateButtons;
end;
```

Ako vidíme, vlastnosť RecordCount nám pomohla odstrániť niektoré nedostatky našej aplikácie. Keďže primárnou funkciou tejto vlastnosti je informovať o počte záznamov, rozhodol som sa tento údaj zahrnúť do hlavičky hlavného okna programu. Na to je potrebná malá modifikácia procedúry UpdateButtnos:

```
procedure TForm1.UpdateButtons;
begin
     if dmAdresar.tabulka.RecordCount=0 then
     begin
        zmaz.Enabled:=false;
        vyprazdni.Enabled:=false;
     end
     else
     begin
        zmaz.Enabled:=true;
        vyprazdni.Enabled:=true;
     end;// if RecordCount=0
                  Form1.Caption:='Adresár,
'+IntToStr(dmAdresar.tabulka.RecordCount)+' záz-
namov';
end;
```

Po odskúšaní programu sa chvíľku môžeme venovať nadšenému jasaniu a potom sa opäť vrátime k seriózne práci. Niekoľkými šikovnými modifikáciami sme síce učinili náš program mierne „inteligentnejším“, k dokonalosti má však ešte ďaleko. Medzi jeho nedostatky patrí napríklad to, že umožňuje pridať aj úplne prázdny záznam. Aby sme tento „neduh“ odstránili, potrebujeme sa oboznámiť s pojmom required field. Tento pojem označuje pole, ktoré musí byť za každých okolností vyplnené, v opačnom prípade komponent TTable nedovolí záznam zapísať do databázy. Pri pokuse o uloženie prázdneho záznamu vznikne výnimka.

V našom prípade bude pre používateľa nevyhnutné, aby vyplnil aspoň prvé tri polia databázy, t. j. priezvisko, meno a adresu. Spustíme teda Database Desktop, otvoríme si našu tabuľku a z menu Tools zvolíme Restructure. Potom vyznačíme pole priezvisko a zaškrtneme políčko Required Field. Podobným spôsobom postupujeme aj v prípade polí meno a adresa. Po kliknutí na tlačidlo Save sa zobrazí nám už dobre známe dialógové okno Restructure Warning (obr. 1). Database Desktop sa nás pýta, či sa majú nové parametre tabuľky aplikovať na už existujúce dáta. V praxi to znamená, že všetky záznamy, ktoré nevyhovejú novým parametrom tabuľky, budú presunuté do súboru Keyviol.db. Ak napríklad máme v tabuľke zapísanú osobu, ktorá nemá vyplnené priezvisko, meno alebo adresu, bude záznam automaticky presunutý do súboru Keyviol.db a z pôvodnej databázy sa odstráni. Po týchto úpravách sa presunieme späť do Delphi, spustíme program a skúsime zapísať neúplný záznam: vyplníme iba priezvisko a meno. Pri pokuse o zápis vznikne výnimka s chybovým hlásením Cannot focus a disabled or invisible window. Po odkliknutí tohto chybového hlásenia sa však aj napriek našej snahe objaví nekorektne vyplnený záznam v databáze. Všetko nasvedčuje tomu, že sme niekde urobili chybu. Pamätáte sa ešte na predošlú časť seriálu? Rozobral som v nej problém duplicity údajov a hovoril som o tom, čo sa stane, keď sa pokúsite zapísať tú istú osobu dvakrát. Poukázal som aj na dôležitosť volania metódy Cancel, ktorá zruší všetky vykonané zmeny. V tomto prípade náš program zobrazuje polovičné údaje práve preto, že sme metódu Cancel nezavolali pred vykonaním implicitnej obsluhy výnimky. Možno vás prekvapil text chybového hlásenia, ktorý sa zobrazil pri pokuse o zapísanie nesprávne vyplneného záznamu. Vedzte, že mňa takisto. Očakával som totiž hlásenie Field value required. Bohužiaľ, aj



Obr. 3

napriek mnohým pokusom sa mi nepodarilo odhaliť príčinu tohto javu. Na svoju obhajobu uvádzam, že podobná procedúra mi fungovala v inej aplikácii bez problémov (t. j. vznikla výnimka s chybovým hlásením Field value required). Budeme teda akceptovať aj toto zvláštne chybové hlásenie, jeho obsah však preložíme po svojom. Pre upokojenie dodávam, že hoci sme nedostali chybovú správu, ktorú som očakával, program funguje korektne. Výnimka vznikne vždy len vtedy, pokiaľ nevyplníme jedno z „required" polí. Upravená verzia „pridávacej" procedúry teda bude vyzerať takto:

```
procedure TForm1.PridajClick(Sender: TObject);
begin
  try
     dmAdresar.tabulka.Append;
     detail.ShowModal;
     if detail.ModalResult=mrOk then dmAdresar.tabulka.Post;
     if detail.ModalResult=mrCancel then dmAdresar.tabulka.Cancel;
  except
     on E:Exception do
     if pos('key violation',AnsiLowerCase(e.message))>0 then
     begin
dmAdresar.tabulka.Cancel;
application.MessageBox('Duplicitné zaradenie!','Chyba',
          MB_OK OR MB_ICONERROR);
      end
      else
     if pos('cannot focus',AnsiLowerCase(e.message))>0 then
     begin
dmAdresar.tabulka.Cancel;
application.MessageBox('Nevyplnili ste pozadované údaje!','Chyba',
          MB_OK OR MB_ICONERROR);
     end
      else
       begin
dmAdresar.tabulka.Cancel;
         raise;
       end;// if pos('key violation' ....
  end;//except
end;
```

Určite ste si všimli, že dosť často využívam funkciu Pos a aj funkciu AnsiLowerCase. Ich použitie som síce rozobral už v prvej časti seriálu, pre istotu to však zopakujem. Vďaka týmto funkciám nemusíme používať doslovné formulácie, pretože budeme pracovať iba s fragmentmi chybových správ. Text celého chybového hlásenia je dostupný pomocou vlastnosti Message objektu E, ktorý je odvodený od objektu Exception. V oboch prípadoch testujeme, či sa v tejto vlastnosti nachádza aspoň časť textu, na základe ktorej určíme, k akému druhu výnimky došlo. Nemusíme teda v porovnávacom výraze vypisovať úplné znenie chybového hlásenia. To však samo osebe nestačí, pretože reťazce nie sú považované za zhodné, keď sa nezačínajú rovnakými písmenami. Preto sme použili funkciu AnsiLowerCase, ktorá nám reťazce skonvertuje na malé písmená. S takto upravenými reťazcami sa potom pracuje oveľa jednoduchšie,

„Klávesománia"

Zmeňme teraz tému a venujme sa „ovládateľnosti" nášho programu. Každá slušná aplikácia Windows by mala byť ovládateľná aj klávesnicou. Ako príklad nám môže poslúžiť okno Environment Options z menu Tools. Takmer každý prvok v tomto okne je prístupný aj pomocou klávesovej skratky: kombináciou Alt +Y sa môžeme presunúť do textového poľa Grid Size Y, kombináciou Alt + B môžeme zaškrtnúť/zrušiť zaškrtnutie políčka Break on Exception atď. Niečo podobné sa dá veľmi jednoducho zrealizovať aj v Delphi.

Vlastnosť FocusControl komponentu TLabel nám umožňuje vytvoriť prepojenie s iným komponentom. Pokiaľ vlastnosť Caption obsahuje tzv. accelerator key, po použití klávesovej skratky sa fokus presunie na komponent určený vlastnosťou FocusControl. Aby komponent TLabel vedel, na ktorú klávesovú skratku má reagovať, musíme umiestniť do vlastnosti Caption znak & (ampersand), a to pred to písmeno, ktoré má v súčinnosti s klávesom Alt vytvoriť klávesovú skratku (po tejto úprave bude príslušné písmeno podčiarknuté). Ak napríklad chceme presunúť fokus na textové pole zobrazujúce krstné meno pomocou skratky Alt + M, musíme v príslušnom komponente TLabel umiestniť pred písmeno M znak &.

Okrem uvedených čŕt majú aplikácie Windows ešte jednu dôležitú vlastnosť: stlačenie klávesu Enter je ekvivalentné stlačeniu tlačidla, ktoré je buď definované ako „defalut button", alebo má práve fokus. Toto tlačidlo vždy bezpečne spoznáte podľa obrysovej čiary, ktorá je hrubšia ako obrysová čiara ostatných tlačidiel. Vezmime si napríklad dialógové okno Uložiť ako... z programu Microsoft Word. Ak sa pozorne prizrieme tlačidlu Uložiť, zistíme, že jeho obrysy sú naozaj trochu hrubšie. Stlačenie klávesu Enter teda bude mať rovnaký efekt ako stlačenie tlačidla Uložiť. Vo väčšine prípadov, ak je na obrazovke prítomné tlačidlo OK, práve toto tlačidlo, bude mať hneď po zobrazení okna hodnosť „default button". Inými slovami, ak nie sú v dialógovom okne vykonané nijaké zmeny, je stlačenie klávesu Enter väčšinou ekvivalentné stlačeniu tlačidla OK.

Možno sa teraz pýtate, prečo som použil „alibistické" spojenie „nijaké zmeny". Príčina je celkom prozaická: môže sa totiž stať, že manipuláciou obsahu okna (nap. pohybom klávesu Tab) sa fokus presunie na úplne iné tlačidlo, napríklad aj na tlačidlo Zrušiť. Do hodnosti „defalut button" bude potom povýšené práve toto tlačidlo. Ukážeme si to na konkrétnom príklade: v programe MS Word si otvoríme dialógové okno Uložiť ako... a stláčame kláves Tab dovtedy, kým kontúry tlačidla Zrušiť nezhrubnú. Sprievodným javom nášho experimentu bude aj objavenie sa focus rectangle na tomto tlačidla. Ak potom stlačíme kláves Enter, súbor nebude uložený, pretože fokus malo tlačidlo Zrušiť, a teda stlačenie klávesu Enter bolo ekvivalentné stlačeniu tohto tlačidla. Našťastie hrubšia obrysová čiara a focus rectangle nám vždy pomôžu identifikovať tlačidlo, ktoré má práve fokus a umožnia nám tak zabrániť neželaným výsledkom.

Ďalšou vlastnosťou programov Windows je možnosť uzavrieť dialógové okno pomocou klávesu Enter. Ako príklad nám opäť poslúži MS Word, konkrétne dialógové okno Tlačiť. Otvoríme ho a stlačíme kláves Esc. Okno sa okamžite zatvorí. Presunieme fokus na tlačidlo Možnosti... a opäť stlačíme Esc. Ako vidíme, okno sa zatvorí vždy a nezávisle od toho, ktorý ovládací prvok má práve fokus.

Niečo podobné sa dá v Delphi zrealizovať veľmi jednoducho: nastavením vlastnosti Default tlačítka dobre na true a nastavením vlastnosti Cancel tlačidla zrus na false. Po týchto úpravách bude stlačenie klávesu Enter ekvivalentné stlačeniu tlačidla dobre a stlačenie klávesu Esc bude ekvivalentné stlačeniu tlačidla zrus.

Okrem uvedených spôsobov ovládania je v aplikáciách Windows štandardom aj pohyb medzi ovládacími prvkami pomocou klávesu Tab. Poradie, v ktorom budete medzi jednotlivými prvkami „cyklovať", je možné nastaviť buď priamo pomocou vlastnosti TabOrder, alebo (a táto cesta je omnoho pohodlnejšia) pomocou dialógového okna Edit Tab Order (obr. 2).

Vyhľadávanie a „zdobenie"

Keď bola naša aplikácia v štádiu „laboratórneho experimentu", obsahovala možnosť hľadania mien. Mechanizmus vyhľadávania však bol však značne nepohodlný, museli sme totiž vždy napísať meno (alebo jeho časť) a potom stlačiť tlačidlo Nájsť. V inovovanej verzii bude vyhľadávanie podobné rýchlemu hľadaniu súborov v NC pomocou klávesu Alt. Výhody takéhoto riešenia sú zrejmé: nemusíme ustavične stláčať tlačidlo Nájsť. Implementácia pre nás už nebude nijakou novinkou, použijeme starú známu metódu Locate. Tentoraz však nebude „úradovať" v obsluhe udalosti OnClick, pretože ju premiestnime do obslužnej rutiny udalosti OnChange komponentu TEdit, ktorý pomenujeme hladaj.

```
procedure TForm1.hladajChange(Sender: TObject);
begin
dmAdresar.tabulka.Locate('priezvisko',hladaj.text,[loPartialKey,loCaseInsensitive]);
end;
```

Na záver si náš program trošku „ozdobíme" a pridáme možnosť vyvolania poštového klienta. Ikony na tlačítkach síce vyzerajú pekne, bez nápisov však tieto tlačidla pôsobia veľmi strohým dojmom. Chybu preto napravíme a každému tlačidlu priradíme zodpovedajúci nápis. Polohu ikonky vzhľadom na nápis určuje vlastnosť Layout. Nastavením tejto vlastnosti na blGlyphTop dosiahneme, že ikonka bude zobrazená nad textom. Okrem toho môžeme tlačidlám pomocou vlastnosti Hint priradiť bublinovú nápoveď. Musíme sa však najprv presvedčiť, či je vlastnosť ShowHint nastavená na true, v opačnom prípade sa totiž „hinty" nezobrazia.

A teraz už sľúbené vyvolanie poštového klienta. Pravdepodobne už väčšina z vás pracovala s internetovými prehliadačmi. Pokiaľ na WWW stránke kliknete na e-mailovú adresu, automaticky sa otvorí poštový klient. Podobne sa správa aj MS Word 97: ak v texte narazí na WWW adresu, podčiarkne ju. Po kliknutí na hypertextový odkaz sa potom spustí internetový prehliadač.

V Delphi sa dá niečo podobné zrealizovať pomocou funkcie ShellExecute. Pred jej použitím však budeme musieť od USES klauzuly unitu hlavného formulára pridať odkaz na jednotku ShellAPI. Potom pridáme na hlavný formulár tlačidlo, ktoré nám bude poštového klienta spúšťať. Obslužná rutina udalosti OnClick tohto tlačidlo má takúto podobu:

```
procedure TForm1.emailClick(Sender: TObject);
var adresa:string;
PDir,parms:array[0..1] of char;
```

```
PAdresa:array[0..30] of char;
begin
adresa:=dmAdresar.tabulka.FieldByName('email').AsString;
   if adresa='' then
   begin
Application.MessageBox('Táto osoba nemá vyplnenú email adresu!',
        'Pozor',MB_OK or MB_ICONEXCLAMATION);
         Exit;
   end;//if if adresa=''
   adresa:='mailto: '+adresa;
   StrPCopy(PAdresa,adresa);
ShellExecute(form1.handle,'open',PAdresa,parms,pdir,sw_shownormal);
end;
```

Pomocou funkcie FieldByName zistíme e-mailovú adresu dotyčnej osoby. Parametrom tejto funkcie je názov poľa databázy, ktorého hodnotu chceme získať. Pokiaľ bude pole email prázdne, vypíše sa upozornenie a vykonávanie procedúry sa ukončí. Keď už máme text e-mailovej adresy, pridáme na jeho začiatok reťazec „mailto: ". Konverziu na reťazec ukončený nulou zabezpečí funkcia StrPCopy. Konečná podoba parametra odovzdávaná funkcii ShellExecute prostredníctvom premennej PAdresa má teda tvar „mailto:" + príslušná e-mailová adresa. Ostatné parametre tejto funkcie si preberieme inokedy, keďže na ich pochopenie sa najskôr treba oboznámiť s reťazcami ukončenými nuou (null terminated strings). Konečný výsledok môjho úsilia vidíte na obrázku č. 3.

V súvislosti s pridaním tlačidla email pripomínam, že toto tlačidlo by malo byť v prípade prázdnej databázy neprístupné (rovnako ako tlačidlá zmaz a vyprazdni). Preto bude potrebné procedúru UpdateButtons mierne upraviť:

```
procedure TForm1.UpdateButtons;
begin
     if dmAdresar.tabulka.RecordCount=0 then
     begin
        zmaz.Enabled:=false;
        vyprazdni.Enabled:=false;
        email.enabled:=false;
     end
     else
     begin
        zmaz.Enabled:=true;
        vyprazdni.Enabled:=true;
        email.enabled:=true;
     end;// if RecordCount=0
    Form1.Caption:='Adresár, '+IntToStr(dmAdresar.tabulka.RecordCount)+'
záznamov';
end;
```

To je nadnes všetko. Domácu úlohu nedostanete, užite si pekné počasie.

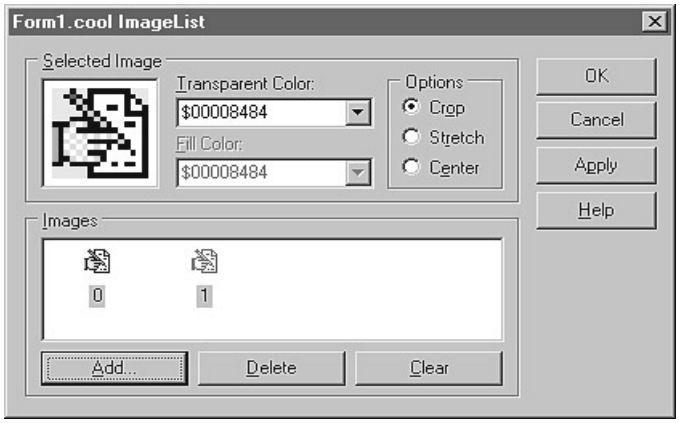
Šiesta časť: Metamorfózy III

Dnes budeme pokračovať v „softvérových reformách". Na úvod jedna nepríjemná správa: Naša aplikácia má (už tradične) jednu chybičku. Program totiž nedovolí zapísať dve osoby s rovnakým menom. Možno si poviete, že tak to predsa má byť, veď chceme predísť duplicite údajov. Môžu sa však vyskytnúť ojedinelé prípady, keď sa dve osoby volajú úplne rovnako. Preto musíme opäť zmeniť štruktúru databázy. Minule sme poradie polí dosť poprehadzovali práve z toho dôvodu, že tretím kľúčovým poľom (key field) bude adresa, ktoré musí nasledovať bezprostredne po poliach priezvisko a meno. Pokiaľ vás výklad o databázach už začínal nudiť, mám pre vás aj jednu dobrú správu: Okrem iného sa dnes budeme venovať aj estetickej stránke nášho programu.

Komponent TToolBar

Väčšina z vás už zrejme videla nástrojový pruh v Internet Exploreri či Nestcape Navigatore. Ten sa na rozdiel od nástrojových pruhov v starších aplikáciách skladá z plochých tlačidiel, ktoré sa stali veľmi populárnymi a tak ich začali používať vo svojich produktoch aj iné firmy vrátane Inprise. Podobný elegantný nástrojový pruh je možné vytvoriť pomocou komponentu TToolBar, ktorý je (našťastie) súčasťou VCL. Musím však podotknúť, že tento komponent je k dispozícii len v Delphi 3 a vyšších verziách. Skôr než sa pustíme do práce, vysvetlíme si použitie komponentu TImageList. Tento komponent plní funkciu akéhosi „skladu" obrázkov. Všetky ikony, ktoré budú zobrazované v tlačidlách na komponente TToolBar, musia byť „uložené" v tomto komponente. Komponent TToolBar upravíme tak, že tlačidlá na ňom umiestnené budú zobrazovať čiernobiele ikonky, ktoré sa budú meniť na farebné, ak ponad ne prejdeme kurzorom myši (podobne ako v Internet Exploreri). Pridáme teda na hlavný formulár dva komponenty TImageList. Prvý z nich bude zobrazovať čiernobiele ikonky (nazveme ho cool) a druhý farebné ikonky (nazveme ho hot). Potom pravým tlačidlom klikneme na komponent cool a z popup menu vyberieme TImageList Editor. Zobrazí sa nám dialógové okno Image List. Stlačíme tlačidlo Add a vyberieme BMP alebo ICO súbor, ktorý sa má v tlačidle zobraziť (tlačidlá vytvoríme neskôr).

Po vybraní súboru nám Delphi oznámi, že obrázok má väčšie rozmery, ako sú rozmery komponentu TImageList. Pod pojmom rozmery komponentu TImageList sa pritom rozumejú



Obr. 1

vlastnosti Width a Height tohto komponentu, ktoré určujú veľkosťobrázka uchovávaného v komponente. Ak teda tieto dve vlastnosti obsahujú napríklad číslo 16, musí byť veľkosť príslušného obrázka 16 x 16 bodov, v opačnom prípade nám Delphi navrhne, aby sme ikonku rozdelili. My proti rozdeleniu ikony nebudeme protestovať, pretože nám príde vhod. Klikneme preto na tlačidlo Yes. Do TImageListu tak pribudnú dve nové ikony (obr. 1).

Možno si teraz niektorí z vás položili otázku, prečo firma Inprise nedodala ikonky v takej forme, aby sme nemuseli „odklikávať" mätúce dialógové okná. Vtip je v tom, že nami vybraný obrázok obsahuje vlastne dve ikonky: prvú pre stav Enabled a druhú pre stav Disabled. Určite ste si všimli, že nastavenie vlastnosti Enabled komponentu TBitBtn na false má za následok zmenu zobrazovanej ikonky (ikona sa zmení na čiernobielu). Komponent TImageList je implicitne nastavený tak, aby akceptoval ikony s rozmermi 16 x 16 bodov (pozri vlastnosti Width a Height), my sme však vybrali ikony s rozmermi 32 x 16 bodov; Delphi nám preto ponúklo možnosť rozdeliť ikonu na dve časti. Keďže komponent cool má obsahovať iba čiernobiele ikonky, farebnú ikonku vymažeme. Podobným spôsobom pridáme zvyšné tri ikony. V prípade komponentu hot sa postup práce líši iba v tom, že namiesto čiernobielych ikon ponecháme farebné. Na tomto mieste musím upozorniť, že je veľmi dôležité, aby ste dodržali poradie obrázkov (t. j. aby prvá čiernobiela ikona v TImageListe cool zodpovedala prvému farebnému obrázku v TImageListe hot atď.). Staré („neploché") tlačidlá vymažeme a na hlavný formulár pridáme komponent TToolBar. Aby mali tlačidlá umiestnené na komponente plochý vzhľad, musíme vlastnosť Flat nastaviť na true. Ďalšími dôležitými vlastnosťami tohto komponentu sú vlastnosti Images a HotImages. Prvá z nich určuje zoznam obrázkov, ktoré sa budú v tlačidlách komponentu TToolBar zobrazovať, keď sa nad ním nenachádza kurzor myši. Nastavíme ju na cool. Druhá obsahuje zoznam ikon, ktoré sa budú zobrazovať, ak sa ponad niektoré z tlačidiel „zablúdi" kurzor myši. Nastavíme ju teda na hot.

Zoznamy ikon sú teda pripravené, ostáva nám už len vytvoriť samotné tlačidlá. Pravým tlačidlom myši klikneme na komponent TToolBar a z popup menu zvolíme New Button. Okamžite po pridaní tlačidla sa v ňom zobrazí ikona. Pomocou vlastnosti ImageIndex vieme určiť, ktorá ikona z príslušného komponentu TImageList sa má v tomto tlačidle zobraziť. Do tlačidla je tiež možné pridať popisný text (najprv však musíme vlastnosť ShowCaptions komponentu TToolBar nastaviť na true). Je veľmi dôležité, aby sme novovytvoreným tlačidlám dali tie isté mená, ako mali tlačidlá, ktoré sme vymazali, t. j. pridaj, zmaz, vyprazdni a email, v opačnom prípade sa vám program nepodarí skompilovať. Podotýkam, že rozmery tlačidiel na komponente TToolBar sa meniť nedajú, pretože sa automaticky prispôsobujú veľkosti zobrazovanej ikony a aj veľkosti popisného textu. Pokiaľ sa vám ikonky zdajú škaredé, môžete ich zväčšiť a upraviť. Na tento účel odporúčam použiť program Icon Forge od firmy Cursor Arts (http://www.cursorarts.com) Mali by ste tiež vedieť, že pokiaľ nastavíte vlastnosť Flat na true, musíte mať nainštalovanú knižnicu COMCTL32.DLL verzie 4.70 a vyššej, v opačnom prípade program nebude fungovať.

Obrázky v databáze

Aby som náš program trochu spestril, rozhodol som sa rozšíriť ho o možnosť uchovávania obrázkov. Databázový systém Paradox umožňuje okrem „obyčajných" polí (ako sú napr.

Obr. 2

polia integer či string) vytvárať aj polia obsahujúce grafiku. Vďaka tomu je možné grafické dáta umiestniť priamo do *.DB súboru, odpadá teda nutnosť použitia externých súborov.

Pridanie „grafického“ poľa sa od pridania „obyčajného“ poľa líši iba tým, že do kolónky Type napíšeme písmeno G (G ako Graphic), pričom veľkosť poľa nenastavujeme. Novovytvorené „grafické“ pole nazveme fotka. Potom na formulár detail pridáme komponent TDBImage, ktorý nám bude fotografie zobrazovať, a nazveme ho image. Nesmieme tiež zabudnúť vyplniť vlastnosti DataSource a DataField. Teraz si možno niektorí z vás položili otázku, akým spôsobom budeme do databázy vkladať obrázky. Veľmi jednoducho – pomocou schránky (clipboard). Komponent TDBImage totiž obsahuje metódu PasteFromClipboard, vďaka ktorej tento problém hravo vyriešime. Dôležitú úlohu pritom hrá nastavenie vlastnosti Stretch. Ak má táto vlastnosť hodnotu true, bude sa komponent snažiť upraviť rozmery obrázka tak, aby presne zodpovedali rozmerom komponentu, a to aj za cenu deformácie. Pokiaľ však táto vlastnosť bude nastavená na false, rozmery obrázka zostanú nezmenené. „Prilepenie“ obrázka zo schránky budeme realizovať prostredníctvom obslužnej rutiny udalosti OnClick nového tlačidla, ktoré nazveme prilepit:

```
procedure Tdetail.priplepitClick(Sender: TObject);
begin
      Image.PasteFromClipboard;
end;
```

Čo však v prípade, ak chceme obrázok vymazať? V prvom rade musíme pridať nové tlačidlo, ktoré nazveme vymaz. Potom jednoducho prepneme tabuľku do „editovacieho“ režimu a poľu fotka priradíme hodnotu null:

```
procedure Tdetail.vymazClick(Sender: TObject);
begin
      dmAdresar.tabulka.Edit;
      dmAdresar.tabulka.FieldByName('Fotka').Value:=null;
end;
```

Výsledok vidíme na obrázku č. 2.

QuickReport

Hoci je naša aplikácia už na relatívne vysokej esteticko-technickej úrovni, stále jej chýba funkcia, ktorá je vo väčšine programov bežná – možnosť tlače údajov. Ako to už v Delphi býva zvykom, aj túto činnosť má na starosti komponent, presnejšie povedané, sada komponentov s názvom QuickReport. Skôr než začneme tieto komponenty používať, stručne si objasníme základné princípy tvorby tlačových zostáv. Zostava vytvorená pomocou komponentov QuickReport sa skladá z pruhov (bands), ktoré rozdeľujú zostavu na niekoľko častí. Každá časť zobrazuje určité údaje, ako napríklad hlavičku či pätu dokumentu, prípadne dáta z databázy. Pomocou pruhov je možné nastaviť rozmery a ďalšie vlastnosti jednotlivých prvkov tlačovej zostavy (napr. či sa má pruh zobrazovať na každej strane dokumentu alebo iba na poslednej strane atď.) Pruhy sú tvorené komponentmi TQRBand. Veľmi dôležitou vlastnosťou týchto komponentov je BandType. Pomocou je možné nastaviť funkciu a tým aj polohu pruhu. Jednotlivé časti zostavy vytvárame postupným pridávaním komponentov TQRBand a vhodným nastavením vlastnosti BandType.

Obr. 3

Základným prvkom každej tlačovej zostavy je hlavička. Hlavičku vytvoríme nastavením vlastnosti BandType na rbTitle. Ďalší pruh bude obsahovať nadpisy stĺpcov, vlastnosť BandType teda nastavíme na rbColumnHeader. Pod nadpismi stĺpcov zvyčajne nasledujú samotné dáta, vlastnosť BandType teda nastavíme na rbDetail. Medzera medzi jednotlivými riadkami v dátovej oblasti (čiže v sekcii detail) je priamo úmerná šírke tohto pruhu, preto tento pruh trochu zúžime. Po dátach nasleduje päta dokumentu, vlastnosť BandType teda nastavíme na rbPageFooter. Základy teda máme za sebou: náš dokument obsahuje hlavičku, dáta a pätu. Aby to nebolo také jednoduché, trošku si to skomplikujeme: na konci dokumentu umiestnime údaj o celkovom počte záznamov v databáze. Je zrejmé, že na tento účel nám nepostačí nastaviť vlastnosť BandType na rbPageFooter, pretože by sa nám tento údaj vytlačil na každú stranu zostavy. Preto budeme musieť pridať nový pruh a vlastnosť BandType nastaviť na rbSummary. Pruh s takýmto nastavením sa vytlačí vždy v dolnej časti poslednej strany tlačovej zostavy. Podotýkam, že vlastnosť BandType môže obsahovať aj iné hodnoty, o tých si však povieme inokedy.

Prejdime teraz od teoretizovania k činom. Samotná tlačová zostava je vlastne obyčajný komponent TForm, na ktorý umiestnime komponent TQuickRep. Vytvoríme teda nový formulár, pridáme komponent TQuickRep a pomenujeme ho report. Podobne ako ostatné

Obr. 4

databázovo orientované komponenty aj komponent TQuickRep musí vedieť, odkiaľ má čerpať údaje, preto nesmieme zabudnúť správne nastaviť vlastnosť DataSet. Okrem toho je dôležitá aj vlastnosť ReportTitle, ktorá určuje, aký text sa bude zobrazovať v stavovom okne tlačiarne počas tlače dokumentu.

Novovytvorený formulár nazveme tlac1 a uložíme pod názvom zostava1. Potom pridáme komponent TQRBand a vlastnosť BandType nastavíme na rbPageHeader. Pomocou niekoľkých komponentov TQRLabel vytvoríme hlavičku, ktorá bude tvorená veľkým nadpisom Adresár osôb. Veľkosť písma nastavíme na 14 bodov a typ písma nastavíme na Bold. Aby bol titulok presne v strede, klikneme naň pravým tlačidlom a zarovnáme ho pomocou voľby Align. V súvislosti s nastavovaním typu písma by som rád poukázal na jednu dôležitú skutočnosť: Delphi implicitne nastavuje znakovú sadu (charset) na Western, a tak sa môže stať, že sa vám diakritika nebude zobrazovať správne. Preto je potrebné znakovú sadu vždy nastaviť na Central European. Môžeme tak urobiť dvoma spôsobmi: nastavením v dialógovom okne Font alebo priamym nastavením podvlastnosti Charset (nájdeme ju po otvorení vlastnosti Font) na hodnotu EASTEUROPE_CHARSET. Pri tvorbe zostáv a aj pri tvorbe aplikácií Delphi sa snažte používať výhradne písma true type (najlepšie Arial). V opačnom prípade sa totiž môže stať, že na vašom počítači sa síce bude diakritika zobrazovať správne, na inom však už nie.

Vráťme sa späť k práci. Pridáme ďalší pruh a vlastnosť BandType nastavíme na rbColumnHeader. Táto sekcia dokumentu bude obsahovať nadpisy stĺpcov, ktoré vytvoríme pomocou komponentov TQRLabel. Písmo nastavíme na 8 bodov, štýl nastavíme na Bold a vytvoríme nadpisy pre polia, ktoré bude naša zostava zobrazovať. Konkrétne ide o polia priezvisko, meno a adresa. Aby dokument nevyzeral jednotvárne, môžeme tento pruh zvýrazniť: otvoríme si vlastnosť Frame a podvlastnosti DrawTop a DrawBottom nastavíme na true. Podotýkam, že počas návrhu je kedykoľvek možné zobraziť si náhľad pomocou popup menu (musíte však mať v Object Inspectore vybraný komponent TQuickRep).

Po týchto úpravách pridáme pruh, ktorý bude zobrazovať samotné údaje z databázy, vlastnosť BandType teda nastavíme na rbDetail. Dáta sú zobrazované pomocou komponentov TQRDBText. Podobne ako ostatné databázové komponenty aj tieto vyžadujú nastavenie zdroja údajov. Na rozdiel od iných databázových komponentov tu však máme namiesto vlastnosti DataSource vlastnosť DataSet. Pole, z ktorého sa majú čerpať údaje, nastavíme pomocou vlastnosti FieldName. Pruh zúžime, aby medzery medzi riadkami neboli pri veľké. Potom vytvoríme pätu strany, vlastnosť BandType teda bude mať hodnotu rbPageFooter. Päta dokumentu zvyčajne obsahuje číslo strany. Na tento účel som použil komponent TQRSysData s vlastnosťou Type nastavenou na qrsPageNumber. Podotýkam, že tento komponent umožňuje zobraziť aj iné údaje (nap. dátum a čas tlače). Na úplný koniec zostavy som sa rozhodol zaradiť údaj o počte záznamov v databáze. Presunieme sa teda na pruh, ktorého vlastnosť BandType má hodnotu rbSumary, a pridáme komponent TQRLabel, ktorý nazveme PocZaznamov. Text (teda vlastnosť Caption) tohto komponentu budeme nastavovať programovo tesne pred tlačou. Návrh zostavy je teda hotový, ostáva nám už len spojazdniť ju. V prvom rade musíme pridať na hlavný formulár nové tlačidlo (nazveme ho tlac) a do komponentov TImageList nové ikonky so symbolmi tlačiarne. Ďalej je potrebné upraviť procedúru UpdateButtons tak, aby bolo tlačidlo tlac neprístupné, pokiaľ nie sú v databáze nijaké záznamy. Upravená procedúra bude mať nasledujúcu podobu:

```
procedure TForm1.UpdateButtons;
begin
      if dmAdresar.tabulka.RecordCount=0 then
      begin
         zmaz.Enabled:=false;
         vyprazdni.Enabled:=false;
         email.enabled:=false;
         tlac.Enabled:=false;
      end
      else
      begin
         zmaz.Enabled:=true;
         vyprazdni.Enabled:=true;
         email.enabled:=true;
         tlac.Enabled:=true;
      end;// if RecordCount=0
      Form1.Caption:='Adresár,
'+IntToStr(dmAdresar.tabulka.RecordCount)+' záznamov';
end;
```

Ďalej je potrebné pred tlačou nastaviť komponent TQRLabel, ktorý má obsahovať počet záznamov v databáze. Potom si pomocou metódy Preview môžeme nechať zobraziť náhľad:

```
procedure TForm1.tlacClick(Sender: TObject);
begin
     tlac1.PocZaznamov.Caption:='Počet záznamov:
'+IntToStr(dmAdresar.tabulka.RecordCount);
     tlac1.report.preview;
end;
```

Ak sme so zostavou spokojní, môžeme dokument vytlačiť pomocou metódy Print. Celá vec však má jeden háčik: dokument sa síce vytlačí, ale čo v prípade, že máme nainštalovaných niekoľko tlačiarní? Riešenie je aj v tomto prípade veľmi jednoduché:

```
procedure TForm1.tlacClick(Sender: TObject);
begin
     tlac1.PocZaznamov.Caption:='Počet záznamov:
'+IntToStr(dmAdresar.tabulka.RecordCount);
     tlac1.report.PrinterSetup;
end;
```

Po týchto úpravách vám program najprv ponúkne zoznam nainštalovaných tlačiarní a umožní zmeniť ich nastavenia. Podotýkam však, že táto metóda nefunguje v niektorých verziách komponentu TQuickRep korektne (po zmene nastavenia tlačiarne nastane výnimka), preto si zaobstarajte verziu 2.0k, s ktorou už žiadne problémy nie sú. Konečnú podobu tlačovej zostavy vidíte na obrázku č. 3. Zostava je teda hotová, má však jednu chybu: neobsahuje ten najkrajší prvok nášho programu - obrázok. Preto vytvoríme ďalšiu zostavu, ktorá na rozdiel od svojej predchodkyne bude zobrazovať údaje iba jednej osoby spolu s fotografiou. Keďže budeme zobrazovať údaje jediného človeka, nepotrebujeme použiť komponent TQRBand. Postačí nám vytvoriť nový formulár a jednoducho naň umiestniť komponent TQuickRep. Na zobrazovanie obrázkov použijeme komponent TQRDBImage, ktorý pracuje podobne ako TDBImage. Ako som už spomínal, jediný rozdiel medzi databázovými komponentmi QuickReport a štandardnými databázovými komponentmi je ten, že v prípade komponentov QuickReport nenastavujeme dátový zdroj (teda nemajú vlastnosť DataSource), ale priamo tabuľku, z ktorej sa budú požadované údaje čerpať. Pridáme teda komponent TQRDBImage, vlastnosť DateSet nastavíme na dmAdresar.Tabulka a vlastnosť DataField na fotka. Rovnako ako komponent TDBImage aj TQRDBImage má vlastnosť Stretch, ktorú nastavíme na false, aby sa nám obrázky nedeformovali. Pre zjednodušenie som formulár upravil tak, že bude zobrazovať iba tri polia databázy: meno, priezvisko, adresu a, samozrejme, fotografiu. Tenký rám okolo nadpisu a fotky je možné vytvoriť pomocou komponentov TQRShape. Výsledok vidíte na obrázku č. 4. Ostáva nám už len pridať tlačidlo, prostredníctvom ktorého sa bude tlač realizovať; nazveme ho TlacDetail. Obslužná rutina udalosti OnClick tohto tlačidla je veľmi podobná tej predchádzajúcej:

```
procedure TForm1.tlacDetailClick(Sender: TObject);
begin
     tlac2.report.PrinterSetup;
end;

Po pridaní nového tlacidla nesmieme zabudnút upravit procedúru
UpdateButtons:

procedure TForm1.UpdateButtons;
begin
     if dmAdresar.tabulka.RecordCount=0 then
     begin
        zmaz.Enabled:=false;
        vyprazdni.Enabled:=false;
        email.enabled:=false;
        tlac.Enabled:=false;
        tlacDetail.Enabled:=false;
     end
     else
     begin
        zmaz.Enabled:=true;
        vyprazdni.Enabled:=true;
        email.enabled:=true;
        tlac.Enabled:=true;
        tlacDetail.Enabled:=true;
     end;// if RecordCount=0
    Form1.Caption:='Adresár, '+IntToStr(dmAdresar.tabulka.RecordCount)+'
záznamov';
end;
```

Na záver ešte musím upozorniť na jednu smutnú skutočnosť, ktorá sa týka komponentov QuickReport verzie 2.0k: bohužiaľ, v tejto verzii nefunguje kopírovanie skupiny objektov. Na prvý pohľad síce všetko funguje správne (t. j. objekty je možné kopírovať), ak však zvolíme voľbu Preview z popup menu, novovytvorené objekty na zostave neuvidíme. To isté platí, aj keď použijeme na zobrazenie náhľadu metódu Preview. Nanešťastie ani Delphi nie je bez chýb a toto je jedna z tých menej vážnych. To je nadnes všetko, užite si pekné počasie a dovidenia nabudúce.

Siedma časť:
Ukladanie stavových informácií

V tejto časti si rozoberieme problematiku ukladania stavových údajov, konkrétne budeme hovoriť o INI súboroch a registračnej databáze (registroch).

INI súbory

Najprv si objasníme význam pojmu stavová informácia. V podstate ide o údaj typu výška či šírka určitého okna, poloha okna a podobne. Ako príklad nám môže poslúžiť MS Word, ktorý si ukladá polohu a rozmery hlavného okna, zoznam naposledy používaných súborov atď. Ak program ukončíme a potom opäť spustíme, objaví sa jeho hlavné okno presne na tom mieste, na ktorom sa nachádzalo pred ukončením programu. Maličkosť, ale veľmi príjemná. Existuje mnoho možností ukladania stavových informácií. V dobách, keď počítačom vládol MS DOS, si každý program ukladal stavové údaje do vlastného inicializačného súboru. Na tomto mieste musím upozorniť, že hoci som použil slovo inicializačného, nemali tieto súbory nič spoločné s INI súbormi používanými vo Windows 3.x. Formát týchto súborov bol rôzny, od prehľadných konfiguračných súborov, aké používal napríklad program MODPLAY, až po „neprehľadné“, ktorých obsah bol zrozumiteľný jedine autorovi programu.

Prvá významná zmena v systéme ukladania stavových informácií prišla so systémom Windows, ktorý zaviedol INI súbory. Tento systém ukladania bol dokonca priamo podporovaný vo Windows API, čo ušetrilo programátorom mnoho času. INI súbory však mali aj jednu veľkú nevýhodu: boli roztrúsené po celom disku. Microsoft však svoju chybu napravil a s príchodom Windows 95 uzrela svetlo sveta (či skôr monitorov) registračná databáza (registre). Vďaka nej sú všetky stavové údaje systému a aplikácií pokope. Registre v podstate plnia tú istú funkciu ako INI súbory. INI súbory z prostredia Windows už takmer úplne vymizli a programy, ktoré ich ešte stále používajú, sú raritou.

My si však aj napriek tomu ukážeme použitie objektu TIniFile, ktorý prácu s INI súbormi umožňuje. Možno sa teraz pýtate, prečo som použil slovo objekt, a nie slovo komponent. Dôvod je jednoduchý: objekt TIniFile totiž nie je potomkom triedy TComponent, nemožno ho teda nazvať komponentom. V tejto súvislosti by som rád poukázal na rozdiely v spôsobe používania komponentov a „nekomponentov“, teda objektov, ktoré nie sú potomkami triedy TComponent. Predtým si však musíme osviežiť niektoré pojmy týkajúce sa OOP. O vlastnostiach, metódach a udalostiach sme hovorili už neraz, nehovorili sme však o dvoch najdôležitejších metódach, ktoré sú jednými z pilierov OOP. Ide o konštruktor (constructor) a deštruktor (destructor). Prvý z nich má na starosti alokovať potrebnú pamäť a vytvoriť inštanciu triedy – objekt. Druhá metóda robí pravý opak: objekt zničí a uvoľní ním obsadenú pamäť.

Aj pokročilí programátori často nesprávne používajú pojmy objekt a trieda, preto si teraz dovolím uviesť ich definíciu, tak ako ju uvádza biblia všetkých Delphi vývojárov, kniha Mistrovství v Delphi 2 od Marca Cantú. Definícia znie: „Trieda je typ definovaný používateľom, ktorý má nejaký stav, reprezentáciu a správanie. Trieda obsahuje určité interné dáta a určité metódy vo forme procedúr alebo funkcií. Trieda zvyčajne charakterizuje vlastnosti a správanie sa niekoľkých veľmi podobných objektov. Programátor používa triedy na usporiadanie zdrojového kódu a prekladač na generovanie aplikácie. Objekt je inštanciou triedy alebo, inými slovami, premenná dátového typu definovaného triedou. Objekty sú skutočné útvary. Počas behu programu objekty zaberajú určitú časť pamäte pre svoju vnútornú reprezentáciu. Vzťah medzi objektom a triedou je rovnaký ako medzi premennou a dátovým typom.“

Teraz sa možno pýtate, prečo sme sa doteraz nestretli s použitím konštruktora a deštruktora, keď je to jeden zo základných prvkov OOP. Odpoveď je jednoduchá: stretli sme sa s nimi, len o tom ešte nevieme. Delphi ich v „zákulisí“ používalo za nás. Doposiaľ sme totiž používali potomkov triedy TComponent, ktorí sú navrhnutí tak, aby ich Delphi dokázalo vytvoriť a zničiť automaticky. Objekt TIniFile však nie je potomkom triedy Tcomponent, a preto budeme musieť zavolať konštruktor aj deštruktor „ručne“. Podobne ako iné metódy aj konštruktory majú svoje parametre. V prípade objektu TIniFile je parametrom názov INI súboru, s ktorým bude inštancia objektu pracovať. Konštruktory všetkých objektov v Delphi nesú názov Create a deštruktory názov Free. Aby sme mohli začať objekt TIniFile používať, musíme do klauzuly USES pridať unit IniFiles, v ktorom je tento objekt implementovaný. Ďalej je potrebné rozhodnúť sa, aké informácie budeme do inicializačného súboru ukladať; pre začiatok nám postačí ukladanie polohy a veľkosti hlavného formulára. Skôr než vytvoríme inštanciu objektu TIniFile, musíme náš nový objekt najprv deklarovať:

```
var
  Form1: TForm1;
  INISubor:TIniFile;
implementation
```

Všimnime si, že som deklaráciu objektu INISubor umiestnil pred sekciu implementation. Urobil som tak z toho dôvodu, aby bol tento objekt „viditeľný“ aj v ostatných unitoch, ktoré používajú unit hlavného formulára. Ak by som ho deklaroval za kľúčovým slovom implementation, bol by „viditeľný“ iba v rámci toho unitu, v ktorom bol deklarovaný.

Nasledujúcim krokom je vytvorenie objektu pomocou konštruktora. Na tento účel som vytvoril obslužnú rutinu udalosti OnCreate hlavného formulára, v ktorej bude

„zrod objektu“ prebiehať. Parametrom konštruktora bude názov INI súboru spolu s cestou. Pre jednoduchosť som sa rozhodol umiestniť inicializačný súbor do toho istého adresára, v ktorom „sídli“ EXE súbor našej aplikácie. Okrem konštruktora bude hrať dôležitú úlohu vlastnosť ExeName komponentu TApplication, ktorá (ako to napovedá už jej názov) vráti názov EXE súboru našej aplikácie spolu s cestou. Keďže pre nás je zaujímavá iba cesta k EXE súboru, vyextrahujeme ju pomocou funkcie ExtractFilePath. Náš inicializačný kód teda bude vyzerať takto:

```
procedure TForm1.FormCreate(Sender: TObject);
var cesta:String;
begin
     cesta:=ExtractFilePath(Application.ExeName);
     cesta:=cesta+'adresar.ini';
     INISubor:=TIniFile.Create(cesta);
end;
```

Je veľmi dôležité, aby sme po vytvorení inicializačnej rutiny nezabudli vytvoriť procedúru, v ktorej bude zavolaný deštruktor. Ak tak neučiníme, program bude síce pracovať správne, po ukončení nám však zanechá nechcené dedičstvo v podobe neuvoľnenej pamäte. Zničenie objektu sa bude realizovať v obslužnej rutine udalosti OnDestroy hlavného formulára:

```
procedure TForm1.FormDestroy(Sender: TObject);
begin
     INISubor.Free;
end;
```

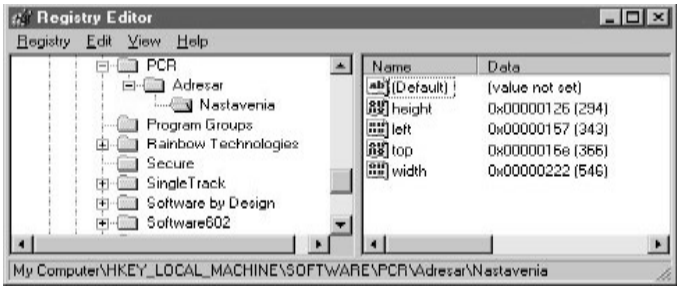
Alokovanie a dealokovanie objektu z pamäte je teda zabezpečené, ostáva nám už len začať ho používať. Veľkosť formulára zistíme pomocou vlastností Width a Height, vlastnosti Left a Top nám zase umožnia zistiť polohu formulára. Vlastnosť Left udáva, o koľko pixlov je ľavý okraj formulára vzdialený od ľavého okraja obrazovky. Podobne vlastnosť Top určuje, o koľko bodov je horný okraj vzdialený od horného okraja obrazovky. Upravená verzia obslužnej rutiny udalosti OnDestroy teda bude vyzerať takto:

```
procedure TForm1.FormDestroy(Sender: TObject);
begin
     with INISubor do
     begin
          WriteInteger('nastavenia','left',Left);
          WriteInteger('nastavenia','top',Top);
          WriteInteger('nastavenia','width',width);
          WriteInteger('nastavenia','height',height);
     end;//with
     INISubor.Free;
end;
```

Ako vidíme, samotný zápis je realizovaný prostredníctvom metódy WriteInteger. Prvým parametrom je tzv. sekcia a druhým tzv. kľúč. Pomocou sekcií je možné INI súbor značne sprehľadniť. Ak napríklad píšeme hudobný program, môžeme vytvoriť sekciu s názvom AudioSettings, ktorá bude obsahovať údaje týkajúce sa nastavení zvuku. Nastavenia týkajúce sa samotného programu potom môžeme umiestniť napr. do sekcie ProgramSettings. Tretím a najdôležitejším parametrom procedúry WriteInteger je samotná hodnota, ktorú chceme zapísať. Aby boli takto uložené údaje na niečo dobré, musíme ich po štarte programu načítať a nastaviť podľa nich príslušné vlastnosti formulára. Na tento účel nám výborne poslúži upravená obslužná rutina udalosti OnCreate:

```
procedure TForm1.FormCreate(Sender: TObject);
var cesta:String;
begin
     cesta:=ExtractFilePath(Application.ExeName);
     cesta:=cesta+'adresar.ini';
     INISubor:=TIniFile.Create(cesta);
     with INISubor do
     begin
          Left:=ReadInteger('nastavenia','left',300);
          top:=ReadInteger('nastavenia','top',300);
          width:=ReadInteger('nastavenia','width',500);
          height:=ReadInteger('nastavenia','height',300);
     end;//with
end;
```

Všimnite si, že nepracujeme s procedúrami, ale s funkciami, ktoré nám vrátia údaje získané z INI súboru. Prvé dva parametre sú pre ReadInterger a WriteInteger zhodné. Tretím parametrom funkcie ReadInteger je tzv. default hodnota. Je to hodnota, ktorú funkcia vráti, ak sa jej z nejakého dôvodu nepodarí prečítať hodnotu zo žiadaného kľúča. To sa nám zíde najmä v tom prípade, pokiaľ náš program spúšťame po prvýkrát, keď INI súbor ešte neexistuje (pretože sa vytvorí až pri ukončovaní programu). Podotýkam, že okrem celých čísel je možné v INI súboroch uchovávať údaje typu boolean (ReadBool/WriteBool) a údaje typu string (ReadString/WriteString).



Obr. 1

Program teda spustíme a trochu sa s ním pohráme. Po krátkom experimentovaní zistíme, že naša aplikácia si síce „zapamätala“ rozmery hlavného formulára, nie však jeho polohu. Príčina tohto javu je celkom triviálna: keď sme vytvárali hlavný formulár, nastavili sme vlastnosť Position na poScreenCenter, v dôsledku čoho bol hlavný formulár vždy automaticky umiestnený do stredu obrazovky, nezávisle od hodnoty vlastností Left a Top. Vlastnosť Position preto nastavíme na poDesigned a všetko bude v poriadku.

Stručný prehľad problematiky INI súborov teda máme za sebou. Skôr než sa začneme venovať registrom, povieme si niečo o objekte TRegIniFile. Použitie tohto objektu je podobné použitiu objektu TIniFile, rozdiel spočíva jedine v tom, že tento objekt nezapisuje údaje do INI súboru, ale do registrov, implicitne do kľúča HKEY_CURRENT_USER. Vývojári firmy Inprise správne predpokladali, že nebude také ľahké upraviť 16 bitové Delphi aplikácie, aby používali Registry, preto do „repertoáru“ VCL zaradili aj tento objekt. Aby sme ho mohli používať, musíme slovo IniFiles v USES klauzule nahradiť slovom Registry a deklaráciu objektu upraviť takto:

```
var
  Form1: TForm1;
  IniSubor:TRegIniFile;
implementation
```

Samozrejme, nesmieme zabudnúť upraviť obslužnú rutinu udalosti OnCreate:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
     INISubor:=TRegIniFile.Create('adresar.ini');
     with INISubor do
     begin
          Left:=ReadInteger('nastavenia','left',300);
          top:=ReadInteger('nastavenia','top',300);
          width:=ReadInteger('nastavenia','width',500);
          height:=ReadInteger('nastavenia','height',300);
     end;//with
end;
```

Keďže sa dáta neukladajú do súboru, ale do registrov, nepotrebujeme už určovať presnú cestu k súboru, preto parametrom konštruktora bude názov INI súboru. Po ukončení aplikácie sa môžete prostredníctvom programu Registry Editor na vlastné oči presvedčiť, že do kľúča HKEY_CURRENT_USER naozaj pribudol podkľúč s názvom adresar.ini a s ďalším podkľúčom s názvom Nastavenia.

Registry

S príchodom Windows 95 prišla na svet nová filozofia ukladania stavových údajov: registre, označované aj ako registračná databáza. Je to vlastne akási centrálna databáza všetkých konfiguračných údajov počítača. Položky registrov sa nazývajú kľúče. Systém Windows obsahuje šesť hlavných kľúčov, sú to HKEY_CLASSES_ROOT, HKEY_CURRENT_USER, HKEY_LOCAL_MACHINE, HKEY_USERS, HKEY_CURRENT_CONFIG a HKEY_DYN_DATA. Kľúč HKEY_CLASSES_ROOT obsahuje pravidlá združovania medzi aplikáciami a typmi súborov (podľa prípony mena súboru). Obsahuje aj informácie na vkladanie a pripojovanie objektov (OLE) združené s COM objektmi a pravidlá združovania podľa tried súborov. Kľúč HKEY_CURRENT_USER obsahuje používateľský profil práve prihláseného používateľa vrátane premenných prostredia, nastavenia pracovnej plochy, sieťových spojení, nastavení tlačiarní a aplikácií. Kľúč HKEY_LOCAL_MACHINE obsahuje informácie o lokálnom počítači vrátane údajov o hrdvéri a operačnom systéme, ako je napr. typ zbernice, systémová pamäť a ovládače. Kľúč HKEY_USERS obsahuje všetky aktívne používateľské profily vrátane profilu HKEY_CURRENT_USER a implicitný profil. Kľúč HKEY_CURRENT_CONFIG obsahuje konfiguračné dáta pre aktuálny hardvérový profil.

Z nášho hľadiska sú najzaujímavejšie kľúče HKEY_CURRENT_USER a HKEY_LOCAL_MACHINE. Keď sa na oba kľúče pozornejšie prizrieme pomocou Registry Editora, zistíme, že obidva obsahujú podkľúč Software. Po otvorení tohto podkľúča uvidíme ďalšie podkľúče, ktoré zväčša nesú názov výrobcu, ktorého softvér je na vašom počítači nainštalovaný. Po otvorení jedného z týchto podkľúčov nájdeme ďalšie podkľúče, pomenované podľa programu, ktorého konfiguračné údaje sú v tomto podkľúči uložené. Ak otvoríme tento podkľúč, dostaneme sa konečne ku konfiguračným položkám určitého

programu. Pripomínam, že nie je povinné ukladať údaje do registrov na základe takejto štruktúry, tieto konvencie však podstatne zlepšujú orientáciu v registroch, a preto by sme ich mali dodržiavať.

Prístup k registrom nám zabezpečuje objekt TRegistry, ktorý sa nachádza v unite Registry (pokiaľ ste si vyskúšali objekt TRegIniFile, nemusíte už pridávať odkaz na tento unit do klauzuly USES). Skôr než si ukážeme použitie objektu TRegistry v praxi, vysvetlíme si filozofiu práce s registrami. Ako som už spomínal, základným prvkom štruktúry registrov je kľúč, pričom existuje šesť koreňových kľúčov (root keys). Pred zápisom do registrov je najskôr potrebné otvoriť kľúč, do ktorého sa budú údaje zapisovať, pričom tento kľúč bude podkľúčom niektorého z koreňových kľúčov. Na nastavenie koreňového kľúča slúži vlastnosť RootKey objektu TRegistry. Postup pri čítaní a zápise bude teda taký, že najskôr nastavíme koreňový kľúč a potom pomocou metódy OpenKey otvoríme niektorý z podkľúčov. Prejdeme teraz od teoretizovania k činom. Najprv vymažeme deklaráciu objektu TRegIniFile a pridáme deklaráciu objektu TRegistry:

```
var
  Form1: TForm1;
  Reg:TRegistry;
implementation
```

Trochu viac práce si budú vyžadovať úpravy v obslužnej rutine udalosti OnCreate. Najskôr treba upraviť volanie konštruktora, ďalej treba nastaviť vlastnosť RootKey, otvoriť kľúč pomocou metódy OpenKey a napokon načítať samotné dáta. Okrem toho je potrebné zaradiť túto obslužnú rutinu do bloku try..except a try..finally, pretože objekt TRegistry pri pokuse o načítanie neexistujúcej položky spôsobí výnimku. Ak sa teda chceme vyhnúť nepríjemnému chybovému hláseniu „Failed to get data for .....“, musíme napísať vhodnú obsluhu výnimky. Upravená verzia inicializačnej rutiny bude vyzerať takto:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
 try
  try
     Reg:=TRegistry.Create;
     with Reg do
     begin
          Rootkey:=HKEY_LOCAL_MACHINE;
          OpenKey('\Software\PCR\Adresar\Nastavenia',true);
          Left:=ReadInteger('left');
          top:=ReadInteger('top');
          width:=ReadInteger('width');
          height:=ReadInteger('height');
     end;//with
  except on E:Exception do
  begin
       if pos('failed to get data',AnsiLowercase(e.message))>0 then
       begin
       end
        else
        begin
            raise;
        end;//else
        end;// if pos...
    end;//except
 finally
    reg.CloseKey;
 end;//finally
 end;
```

Ako vidíme, obslužná rutina najprv zisťuje, či chybový text výnimky neobsahuje uvedené hlásenie. Ak áno, program jednoducho neurobí nič a bude pokračovať ďalej. Všimnite si metódu OpenKey. Jej prvým parametrom je kľúč, ktorý sa má otvoriť. Druhý parameter je oveľa zaujímavejší: ak je nastavený na true, umožňuje vytvoriť kľúč špecifikovaný prvým parametrom, ak ten ešte neexistuje. Keď spustíme program po prvýkrát, metóda OpenKey nám náš kľúč vytvorí automaticky, nemusíme teda použiť metódu CreateKey. Aby po vzniku výnimky neostal kľúč otvorený, zatvoríme ho pomocou metódy CloseKey v rámci bloku try..finally. Samozrejme, rutinu na ukladanie dát tiež treba prepracovať:

```
procedure TForm1.FormDestroy(Sender: TObject);
begin
try
     with Reg do
     begin
          OpenKey('\Software\PCR\Adresar\Nastavenia',true);
          WriteInteger('left',Left);
          WriteInteger('top',Top);
          WriteInteger('width',width);
          WriteInteger('height',height);
     end;//with
 finally
     reg.CloseKey;
 end;//finally
  reg.Free;
end;
```

Aj v tomto prípade som umiestnil metódu CloseKey do bloku try...finally pre prípad, že by počas zápisu do registrov nastala nejaká výnimka. Základy práce s registrami máme teda za sebou. Nasledovať budú už iba malé kozmetické úpravy.

Aby sme náš program sprehľadnili, rozhodol som sa premiestniť všetok kód spojený so zápisom a ukladaním do dvoch procedúr: SaveSetup a LoadSetup. Ich „telá“ budú identické s obslužnými rutinami udalostí OnCreate a OnDestroy, preto ich nebudem vypisovať ešte raz. Jediný rozdiel bude v deklarácii procedúry:

```
procedure TForm1.LoadSetup;
... nasleduje telo procedúry
procedure TForm1.SaveSetup;
... nasleduje telo procedúry
```

Nesmieme zabudnúť ani deklarovať tieto procedúry v sekcii type:

```
type
  TForm1 = class(TForm)
    DBGrid1: TDBGrid;
    Panel1: TPanel;
    hladaj: TEdit;
    Label1: TLabel;
.....................................
procedure SaveSetup;
procedure LoadSetup;
   private
    { Private declarations }
  public
    { Public declarations }
  end;
```

Možno sa pýtate, prečo som kód premiestnil do nových procedúr. Dôvod je jednoduchý: väčšia prehľadnosť. Ak by sme sa z nejakého dôvodu rozhodli proces zápisu zrealizovať inde ako v obslužnej rutine udalosti OnDestroy, museli by sme presunúť do novej procedúry celý jej kód vrátane deklarácie premenných. Takto nám postačí premiestniť iba jediný riadok – volanie procedúry SaveSetup (alebo LoadSetup v prípade načítavania). Upravené obslužné rutiny udalostí OnCreate a OnDestroy budú mať teda nasledujúcu podobu:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
     LoadSetup;
end;
procedure TForm1.FormDestroy(Sender: TObject);
begin
     SaveSetup;
end;
```

Ak ste zvedaví, ako to všetko vyzerá na „mieste činu“, teda v registroch, môžete si konfiguračné údaje prezrieť pomocou Registry Editora (obr. 1). Začiatočníkov však musím upozorniť, aby tento nástroj používali opatrne, pretože neodbornou manipuláciou možno ohroziť chod programov alebo aj celého systému Windows.

To je nadnes všetko, dovidenia nabudúce.

Ôsma časť: Inštalácia

V tejto časti si ukážeme, ako vytvoriť inštaláciu pre našu aplikáciu. Ešte predtým sa však na chvíľku vrátime k téme predchádzajúcej časti seriálu: k registrom.

Problémy s registrami

Počas práce na jednom programe som narazil na určitú nekompatibilitu medzi Windows 95 a Windows NT Workstation 4.0 (bohužiaľ, nemal som možnosť program otestovať pod Windows 98). Ako všetci vieme, metóda DeleteKey objektu TRegistry slúži na vymazávanie kľúčov. Problém je v tom, že táto metóda sa pod Windows 95 a Windows NT Workstation 4.0 (build 1381, Service pack 3) správa odlišne. Pokiaľ máte prístup k obom systémom, môžete si tento problém experimentálne overiť. Nasledujúci kód bude fungovať pod Windows NT, nie však pod Windows 95:

```
procedure TForm1.Button5Click(Sender: TObject);
var reg:TRegistry;
ok:boolean;
begin
   try
    reg:=TRegistry.Create;
```

```
    reg.RootKey:=HKey_Local_machine;
    ok:=Reg.DeleteKey('\Software\TestKey\');
    if ok then showmessage('Kľúč bol úspešne vymazaný');
   finally
     reg.free;
   end;
end;
```

Pod Windows NT bude všetko pracovať bezchybne, pod Windows 95 však, bohužiaľ, nie. Našťastie riešenie je jednoduché: na konci registry kľúča nesmie byť uvedená spätná lomka. „Inkriminovaný" riadok by teda mal vyzerať takto:

```
ok:=Reg.DeleteKey('\Software\TestKey');
```

Pokiaľ bude registry kľúč uvedený bez spätnej lomky, vymazávanie bude fungovať spoľahlivo na oboch systémoch.

Inštalácia

V časoch, keď počítačom vládli prvé verzie MS DOS-u, stačilo program jednoducho prekopírovať z diskety na pevný disk a spustiť. Často sa však stávalo, že program vyžadoval modifikácie súborov AUTOEXEC.BAT či CONFIG.SYS. Aby používatelia tieto zmeny nemuseli robiť ručne, začali výrobcovia vytvárať inštalačné programy, ktoré okrem kopírovania dát vykonávali aj potrebné zmeny v systémových súboroch. Vo väčšine prípadov program prekopíroval svoje dáta do jediného adresára, a tak deinštalácia pozostávala len z vymazania tohto adresára a obnovy súborov AUTOEXEC.BAT či CONFIG.SYS.

S príchodom systému Windows sa však situácia zmenila. Programy kopírovali svoje dáta nielen do svojho vlastného adresára, ale aj do systémového adresára MS Windows. Okrem toho modifikovali súbory WIN.INI a SYSTEM.INI. Pokiaľ výrobca nedodal deinštalačný program, bolo takmer nemožné produkt úplne odinštalovať. Mohli ste síce zmazať adresár, v ktorom dotyčný program „sídlil", to však ani zďaleka neznamenalo, že ste ho odinštalovali – program po sebe mohol zanechať „smeti" v podobe DLL knižníc v systémovom adresári Windows.

Vo Windows 95 a NT je inštalačný a deinštalačný program samozrejmosťou. Jednoducho navštívite menu Pridať/Ubrať programy (Add/Remove Programs), vyznačíte príslušnú aplikáciu a tá sa vzápätí odinštaluje. Podotýkam, že Windows nevymazáva súbory „na vlastnú päsť"; stlačením tlačidla Pridať/Ubrať iba aktivujete deinštalačnú utilitu, ktorú dodal výrobca vašej aplikácie. Treba však podotknúť, že hoci takéto riešenie vyzerá dokonalé, má aj svoje „muchy". Keďže systém Windows umožňuje zdieľať súbory, môže sa veľmi ľahko stať, že deinštalačný program jednoducho nebude vedieť vymazať určitý súbor, pretože ho používa iná aplikácia. Okrem toho sa stáva aj to, že podaktoré programy po sebe síce vymažú všetky súbory, nevymažú však položky z registračnej databázy (registrov).

Inštalačný program je v dnešnej dobe nevyhnutnosťou. Nie každému sa však chce strácať čas jeho písaním, preto už v časoch MS DOS-u existovali rôzne sharewarové utility, ktoré dokázali inštaláciu výrazne zjednodušiť. Azda najpoužívanejším inštalačným programom sa stal program InstallShield firmy Stirling Technologies. Firma ho ponúka v rôznych verziách (InstallShield Express, InstallShield Professional, InstallShield International, InstallShield Java Edition, InstallShield for Windows CE). My budeme používať tú najjednoduchšiu verziu – InstallShield Express, ktorá dodáva s Delphi.

Skôr než začneme pracovať s programom InstallShield, povieme si niečo o „zákulisnej" činnosti Delphi. V prvom rade je potrebné vedieť, že vaše programy (napísané či už v Delphi, alebo inom vývojovom nástroji) nebudú vždy fungovať, pokiaľ ich na iný počítač jednoducho prekopírujete. Dôvodov môže byť mnoho. Treba si uvedomiť, že Delphi robí pomerne veľa vecí za vás v „zákulisí" a pracuje so súbormi, o ktorých existencii nemáte ani potuchy. Ak váš program prekopírujete na iný počítač bez týchto súborov, nebude pracovať správne alebo sa nespustí vôbec. Aby bola situácia ešte zložitejšia, program nemusí fungovať ani po prekopírovaní týchto súborov „ potrebných na prežitie ". Možno sa vám už stalo, že ste chceli s kamarátom vyskúšať chat program, ktorý je dodávaný s Delphi ako demo aplikácia. Na vašom počítači (na ktorom máte nainštalované Delphi) program fungoval, na kamarátovom sa však ani nespustil. Tí zdatnejší z vás možno prišli na to, že komponent TCP (s ktorým tento program pracuje) je vlastne ActiveX komponent, vypátrali príslušný OCX súbor a prekopírovali ho do systémového adresára Windows. Žiaľ, program stále nefungoval. V tomto prípade je problém v tom, že na registráciu komponentov ActiveX do systému je potrebná špeciálna rutina, súbory nestačí jednoducho skopírovať.

Pri vytváraní inštalácie pre program Adresár budeme postavení pred podobný problém. S naším programom musíme dodať aj súbory, s ktorými Delphi aktívne spolupracuje. Aby som bol konkrétnejší, ide o súbory BDE – Borland Database Engine. Je to vlastne súbor DLL knižníc, ktoré poskytujú funkcie na prácu s databázou. Skratka BDE bude určite známa programátorom, ktorí používali Borland Pascal či Borland C++. Vtedy totiž iná cesta k databázam neexistovala. Práca s rutinami BDE je pomerne zložitá, preto sa vývojári z Borlandu (teraz Inprise) rozhodli vytvoriť komponenty, ktoré prácu s databázami podstatne zjednodušujú (tieto komponenty však tiež pristupujú k databázam cez BDE). Pokiaľ vlastníte verziu Delphi Professional, môžete sa pozrieť na zdrojové texty komponentu TTable a uvidíte, ako vyzerá použitie BDE v praxi. Ako som už naznačil, na cieľový počítač treba prekopírovať súbory BDE. InstallShield je našťastie na túto úlohu dobre pripravený a inštaláciu BDE zvláda veľmi dobre. Nielenže náš program (vrátane BDE) nainštaluje, ale ponúkne aj možnosť deinštalácie.

Sama inštalácia BDE na cieľový počítač však ešte nestačí, náš program totiž pracuje s aliasom, ktorý je potrebné vytvoriť ešte predtým, než sa vykonajú akékoľvek operácie, ktoré pracujú s databázou. Preto som sa rozhodol umiestniť rutinu na vytvorenie aliasu do obslužnej rutiny udalosti OnCreate hlavného formulára:

```
procedure TForm1.FormCreate(Sender: TObject);
var Aliases:TStrings;
DataDir:String;
begin
 try
  Aliases := TStringList.Create;
  Session.GetAliasNames(aliases);
  if Aliases.IndexOf('adresar')=-1
  then
  begin
    DataDir:=ExtractFilePath(application.exename)+'data';
    Session.AddStandardAlias('adresar',DataDir,'PARADOX');
    Session.SaveConfigFile;
 end;{if  aliases.indexOf('adresar')=-1}
 finally
  Aliases.Free;
 end;
  LoadSetup;
end;
```

Najskôr treba zistiť, či daný alias v systéme už existuje. Na to je potrebné zistiť mená všetkých aliasov, ktoré sú v systéme použité. Na tento účel slúži metóda GetAliasNames komponentu TSession. Možno si teraz položíte otázku, odkiaľ som nabral komponent TSession, keď sme ho nikde nedeklarovali a nikdy sme ho ani nespomínali. Vysvetlenie je jednoduché: každý databázový program napísaný v Delphi potrebuje na svoju prácu najmenej jeden komponent TSession, a preto sa tento komponent vytvorí v „zákulisí" automaticky. Pomocou metódy IndexOf zisťujeme, či je daný alias (teda reťazec „adresar") prvkom objektu TStringList. Pokiaľ objekt takýto reťazce naozaj obsahuje, vráti nám jeho index, v opačnom prípade vráti –1. Inými slovami, pokiaľ náš alias ešte nebol vytvorený, funkcia IndexOf nám vráti hodnotu –1.

Adresár som navrhol tak, aby sa dáta nachádzali vždy v podadresári adresára, v ktorom je umiestnený EXE súbor. Pokiaľ teda bude program nainštalovaný v adresári C:\Pokus, databázové súbory sa budú nachádzať v adresári C:\Pokus\Data. Ďalší riadok pomocou funkcie ExtractFilePath zisťuje názov adresára, v ktorom sa práve nachádza EXE súbor nášho programu. Potom sa k tomuto názvu pridá reťazec „data", čím vlastne dostaneme úplnú cestu k adresáru, v ktorom sa nachádzajú databázové súbory. Takže cestu už máme, teraz už len vytvoriť alias. Na tento účel použijeme metódu AddStandardAlias komponentu TSession.

Poslednou dôležitou metódou je SaveConfingFile, ktorá zabezpečí uloženie aliasu do konfiguračného súboru BDE. Ak by sme ju nepoužili, alias by bol pre ostatné BDE aplikácie viditeľný iba dovtedy, kým je BDE v pamäti. Aby som bol názornejší, demonštrujem vám celý problém na jednoduchom pokuse: Pomocou Database Desktopu vymažte alias adresar, potom vymažte riadok Session.SaveConfigFile a program spustite. Bude pracovať správne, a pokiaľ otvoríte Database Desktop, náš alias bude na svojom mieste. Aby sa výsledný efekt prejavil, musíme ukončiť všetky aplikácie používajúce BDE, preto zavrieme Delphi aj Database Desktop. Potom Database Desktop opäť spustíme a zistíme, že náš alias zmizol. Ak spustíme Delphi, dostaneme chybové hlásenie „Unkown alias", vlastnosť Active komponentu Tabulka sa automaticky nastaví na false a nepodarí sa nám ju opäť nastaviť na true (keďže náš alias neexistuje). Program teda bez aliasu nebude vôbec fungovať. Preto Delphi ukončíme a alias pridáme opäť ručne (pomocou Database Desktopu). Len čo bude alias späť na svojom mieste, Delphi otvorí náš projekt korektne (nezabudnite nastaviť vlastnosť Active komponentu Tabulka na true). Potom riadok so Session.SaveConfingFile vrátime na svoje miesto. Presunieme sa do Database Desktopu a alias opäť vymažeme. Potom Database Desktop ukončíme a spustíme náš program. Alias je pridaný automaticky, ostáva nám už len overiť, či nezmizne po zatvorení všetkých programov používajúcich BDE. Ukončíme preto Delphi a spustíme Database Desktop. Ak ste správne postupovali, alias je na svojom mieste.

InstallShield Express

Problém s aliasom sme vyriešili, môžeme teda prejsť k opisu programu InstallShield Express. Vzhľad tohto programu je mierne odlišný od vzhľadu bežných aplikácií Windows. V nasledujúcich odsekoch si rozoberieme všetky voľby programu.

Set visual design

Voľby združené pod touto hlavičkou ovplyvnia vizuálnu stránku inštalácie. Máte napríklad možnosť umiestniť do pozadia bitmapu, nastaviť názov aplikácie, ktorý sa

bude počas inštalácie zobrazovať v ľavom hornom rohu obrazovky, atď. V tejto chvíli je pre nás dôležité, aby bola zaškrtnutá voľba Automatic unistaller na palete Features.

Select InstallShield Objects for Delphi
Toto je azda najdôležitejšia voľba, pretože sa týka BDE a jeho konfigurácie. Kliknite na tlačidlo vedľa General Options a objaví sa vám dialógové okno, v ktorom zaškrtnite voľbu BDE (Borland Database Engine). Objaví sa vám ďalšie dialógové okno: BDE Installation Type. Môžete si vybrať medzi plnou (Full BDE Installation) a čiastočnou (Partial BDE Installation) inštaláciou. Výhodou čiastočnej inštalácie je to, že zaberá menej miesta, nevýhodou to, že pokiaľ zvolíte tento typ inštalácie, používateľ nebude môcť na svojom počítači spustiť viac ako jednu BDE aplikáciu. Pokiaľ chcete presne vedieť, z akých súborov BDE pozostáva, kliknite na tlačidlo Help. V texte nápovede nájdete odkaz na Partial BDE Installation. Kliknite na tento odkaz a dostanete sa k vyčerpávajúcemu opisu všetkých súborov BDE.

Vráťme sa však späť k inštalácii BDE. Zvolíme Full BDE Installation a stlačíme tlačidlo Next. Objaví sa dialógové okno BDE Alias Step 1 of 4. Keďže sme si vytvorenie aliasu zabezpečili priamo v našom programe, môžeme všetky štyri dialógové okná bez povšimnutia „odkliknúť". Potom sa opäť ocitneme pred dialógovým oknom Select InstallShield Objects for Delphi. Klikneme na OK a sme naspäť v hlavnom menu.

Specify Components and Files
Skupina volieb pod touto hlavičkou vám umožňuje špecifikovať, ktoré súbory sa majú kam inštalovať. Používateľ má možnosť vybrať si jednu z nasledujúcich typov inštalácií:

- **Typická** (Typical) zvyčajne zahŕňa všetky súbory aplikácie.
- **Kompaktná** (Compact) zahŕňa iba tie súbory, ktoré sú nevyhnutné na beh programu.
- **Používateľská** (Custom) umožní používateľovi vybrať si, ktoré skupiny súborov sa budú inštalovať.

Aby sme tieto tri typy inštalácie mohli používať, musíme najskôr stlačiť tlačidlo vedľa nápisu Dialog Boxes (skupina Select User Interface Components) a zaškrtnúť voľbu Custom Setup. Náš program v podstate pozostáva z dvoch skupín súborov: program samotný (t. j. EXE súbor a knižnica Data s databázami) a BDE. Pokiaľ teda umožníme používateľovi zvoliť si typ inštalácie, dáme mu na výber dve možnosti: ak na svojom počítači ešte nemá nainštalovaný BDE (čo je dosť pravdepodobné), môže použiť voľbu Typical; na jeho stroj bude nainštalovaný BDE spolu s naším programom. Pokiaľ používateľ už BDE nainštalovaný má (napríklad tiež pracuje s Delphi), môže použiť voľbu Compact; na jeho počítač sa tak nainštaluje iba náš program s databázami (bez BDE). Predpokladám, že použitie voľby Custom netreba podrobne rozoberať.

InstallShield Express pracuje s tzv. skupinami súborov – groups. Každá skupina súborov má presne určený adresár, do ktorého sa súbory prekopírujú. Základnou skupinou je skupina Program Files (nemýliť si s adresárom Program Files!), do ktorej zaradíme najzákladnejší prvok nášho programu – EXE súbor. Zaradenie je veľmi jednoduché: spustíme Explorer a súbor jednoducho „pretiahneme" do skupiny Program Files. Všimnite si položku Destination directory na pravej strane dialógového okna. V editačnom okne je nápis <INSTALLDIR>. Znamená to, že súbory skupiny Program Files sa budú inštalovať do toho adresára, ktorý zadá používateľ (musí však byť zaškrtnutá voľba Choose Destination Location v dialógovom okne Dialog Boxes, pozri skupinu Select User Interface Components). My však potrebujeme nainštalovať nielen EXE súbor, ale aj databázy. Keďže tie sa nebudú inštalovať do toho istého adresára ako program, ale do jeho podadresára, musíme pomocou tlačidla Add group vytvoriť novú skupinu. Objaví sa dialógové okno Add group. Keďže táto skupina bude reprezentovať databázové súbory, nazval som ju Data files. Súbory tejto skupiny sa budú inštalovať do podadresára Data. Tento podadresár bude vytvorený v adresári, ktorý zadá používateľ. Vyjadrené v jazyku InstallShieldu, adresár Data bude vytvorený v adresári <INSTALLDIR>\Data. Potom sa presunieme do Explorera, vyberieme databázové súbory a „pretiahneme" ich do skupiny Data. Časť práce teda máme za sebou.

Ďalším dôležitým prvkom inštalácie sú komponenty aplikácie alebo aplikačné komponenty (application components). Ako som už spomínal, InstallShield Express umožňuje vytvárať tri základné typy inštalácií. Pri každom type je potrebné určiť, ktoré súbory sa budú na cieľový disk inštalovať. Aby nás autori InstallShieldu ušetrili nadbytočnej práce so súbormi, zaviedli použitie skupín (groups). Pri špecifikácii jednotlivých druhov „setupov" však nepracujeme so skupinami, ale s aplikačnými komponentmi (application components). A budú to práve ich názvy, ktoré bude používateľ vidieť, keď si zvolí používateľskú inštaláciu. Aplikačné komponenty združujú skupiny (groups), podobne ako skupiny združujú súbory. V našom prípade sme program rozdelili na dve základné skupiny: adresár osôb a BDE. Program samotný sme ešte rozdelili do dvoch skupín: program a dáta. V dialógovom okne Specify Components and Files máme možnosť priradiť skupiny príslušným aplikačným komponentom. Do kategórie Application Files budú spadať skupiny Program Files a Data Files (keďže program a dáta sú od seba neoddeliteľné, musia byť súčasťou toho istého aplikačného komponentu). Skupinu priradíme určitému aplikačnému komponentu tak, že ho najprv vyznačíme a potom stlačíme tlačidlo Add to Application Component. Ďalší aplikačný komponent bude združovať skupiny okolo BDE, preto som ho nazval BDE Files. Tento komponent musí obsahovať nasledujúce skupiny: BDE/IDAPI Files, BDE/IDAPI BLL Files a BDE/IDAPI CNF File.

Aplikačné komponenty teda máme vytvorené, ostáva nám už len priradiť ich jednotlivým typom inštalácií. V zozname Setup Types najprv zvolíme typ „setupu", presunieme sa do zoznamu Application Components (na pravej strane okna), vyznačíme príslušnú položku a stlačíme Add to Setup Type. Používateľská inštalácia (Custom) bude obsahovať oba aplikačné komponenty (BDE Files aj Application Files). Typická inštalácia (Typical) bude tiež obsahovať oba aplikačné komponenty, keďže táto voľba je určená pre typického používateľa a ten na svojom počítači pravdepodobne BDE nemá nainštalovaný. Posledný typ inštalácie, kompaktná (Compact), by mal obsahovať iba minimum súborov potrebných na beh programu, preto som doň zaradil iba aplikačný komponent Application Files.

Select User Interface Options
Po stlačení tlačidla Dialog Boxes sa objaví dialógové okno, ktoré vám umožní nastaviť vzhľad a správanie inštalačného programu počas procesu inštalácie. Práca s týmto oknom je veľmi jednoduchá a nebudem sa jej hlbšie venovať.

Make Registry Changes
Keďže registry kľúče si náš program overuje a vytvára automaticky, netreba sa touto položkou zaoberať. Nevýhodou takéhoto prístupu je, že InstallShield Express o našich registry kľúčoch „nevie", a teda pri deinštalácii ich neodstráni.

Specify Folders and Icons
Po stlačení tlačidla General Settings či Advanced Settings sa objaví dialógové okno, ktorého funkcia je zrejmá už na prvý pohľad, preto ho tiež nebudem rozoberať.

Run Disk Builder
Po stlačení tlačidla Disk Builder sa vygenerujú inštalačné diskety. Proces generovania je časovo dosť náročný a môže trvať aj niekoľko minút. Počas vytvárania diskiet vás bude InstallShield Express prehľadne informovať o tom, aká operácia práve prebieha, a oboznámi vás aj s prípadnými chybami, ktoré nastali počas generovania. Okrem chýb sa zobrazujú aj varovania (warnings). Rozdiel medzi varovaniami a chybami je v tom, že chyba spôsobí prerušenie vytvárania inštalačných diskiet (ak napr. chýba nejaký súbor). Varovanie vás iba upozorní, že ste na niečo pozabudli (napr. že ste zabudli vyplniť názov aplikácie), proces generovania sa nepreruší.

Test the Installation
Pomocou voľby Test Run si môžete otestovať funkčnosť vašej inštalácie. Túto voľbu odporúčam použiť len pri prvej sérii testov, keď upravujete vzhľad a správanie inštalačného programu. Na seriózne testovanie je nevyhnutné program preniesť na „čistý" počítač, na ktorom predtým nebolo nainštalované ani Delphi, ani BDE. Pokiaľ na počítači Delphi či BDE nainštalované boli a neskôr boli odinštalované, presvedčte sa, že po nich neostali nijaké „smeti": Prehľadajte adresár Program Files a skúste nájsťadresár BDE. Pokiaľ existuje, vymažte ho. Potom hľadajte v registroch slovo Borland. Vymažte všetky kľúče, v ktorých sa toto slovo nachádza a ktoré nejako súvisia s BDE. Nemusím pripomínať, že do registrov by menej skúsení používatelia nemali zasahovať, pretože môžu narobiť viac škody ako úžitku.

Create Distribution Media
Voľbu Copy to Floppy nepoužívajte. Sám som ju použil iba raz a mal som z nej dosť nepríjemný pocit. Síce funguje, no kopírovanie trvá neúnosne dlho. Najlepšie urobíte, ak inštalačné adresáre vytvorené InstallShieldom jednoducho skopírujete na diskety „ručne".

Na záver by som chcel podotknúť, že správne vyladenie inštalácie je časovo dosť náročný proces, preto sa po prvých nezdaroch nevzdávajte. Musíte pomerne veľa experimentovať, na mnohé veci prídete jedine metódou pokus – omyl. Želám vám veľa trpezlivosti pri vytváraní inštalácií a teším sa na ďalšie stretnutie.

Deviata časť: SQL

V tejto časti si povieme niečo o jazyku SQL. Najskôr si ukážeme niekoľko jednoduchých príkladov, ktoré vám pomôžu osvojiť si základy tohto jazyka, a potom si rozoberieme použitie komponentu TQuery, ktorý umožňuje použitie SQL dopytov v Delphi. Podotýkam, že v tomto článku sa budeme zaoberať s SQL výhradne v súvislosti s lokálnymi databázami (dBase a Paradox), preto nebudem rozoberať pojmy ako triggers a stored procedures.

Trocha teórie
Skratka SQL znamená Structured Query Language, čo by sme mohli veľmi voľne preložiť ako štruktúrovaný dopytovací jazyk. Základná schéma práce s jazykom SQL je takáto: pomocou jazyka SQL sformulujete príkaz (SQL query, po slovensky SQL

dopyt), ktorý má príslušný databázový stroj (database engine) vykonať. Odpoveďou počítača bude množina údajov, ktorá vyhovuje kritériám špecifikovaným v SQL dopyte (pokiaľ použijete príkaz `Select`; samozrejme, pokiaľ použijete príkazy ako `Update`, `Insert` či `Delete`, výsledkom nebude množina údajov, pretože tieto príkazy slúžia na manipuláciu s dátami, a nie na výber dát). Je potrebné vedieť, že jazyk SQL vám umožní špecifikovať, ktoré údaje z databázy má počítač vybrať, neumožňuje vám však počítaču „povedať“, ako sa má príslušný SQL dopyt spracovať; preto je nevyhnutné, aby bol databázový systém dobre navrhnutý.

Ako je zrejmé z uvedeného textu, hlavnými príkazmi jazyka `SQL` sú `Select`, `Update`, `Insert` a `Delete`. My sa zameriame na príkaz `Select`, pretože manipulácia údajov sa dá zvládnuť aj bez použitia SQL; postačia nám metódy komponentu TTable. Pri výbere a triedení údajov je však príkaz `Select` neoceniteľným pomocníkom: pomocou niekoľkých riadkov SQL zvládneme to, čo by sme pomocou filtrovacích funkcií komponentu TTable pravdepodobne nezvládli vôbec.

Teória v praxi

Ako základná pomôcka pri výučbe jazyka SQL nám bude slúžiť program Database Explorer. Ako zdroj údajov použijeme niekoľko demo – databáz, ktoré sú dodávané s Delphi. Najzákladnejším príkazom jazyka SQL je príkaz `Select`. Znalci cudzích jazykov vedia, že slovo select znamená po anglicky vybrať, tento príkaz teda bude slúžiť na výber určitých údajov z databázy. Náš prvý SQL dopyt bude čerpať údaje z tabuľky COUNTRY.DB:

```
select * from country
```

Po vykonaní tohto príkazu nám Database Explorer zobrazí kompletný výpis tabuľky COUNTRY.DB. Príkaz `Select` už poznáme, a tak je načase vysvetliť, čo znamená tá záhadná hviezdička. Tento špeciálny znak povie databázovému stroju, aby do výslednej množiny dát zahrnul všetky polia z databázy. Predstavme si však, že nás zaujíma iba názov krajiny a jej hlavné mesto. SQL dopyt bude v tomto prípade vyzerať takto:

```
select name, capital from country
```

Teraz si možno položíte otázku, čo v prípade, ak chceme pracovať iba so štátmi Severnej Ameriky. Pomocnú ruku nám podá klauzula `Where`, ktorá v spolupráci s príkazom `Select` umožňuje vybrať z databázy len tú množinu dát, ktorá spĺňa určité kritériá. SQL dopyt bude mať nasledujúcu podobu:

```
select * from country where continent = „North America“
```

Jazyk SQL sám osebe nie je „case sensitive“, čiže nerozlišuje veľké a malé písmená; no výraz, ktorý sa nachádza v úvodzovkách, veľké a malé písmená rozlišuje. V prípade, že napíšeme SQL dopyt

```
select * from country where continent = „North america“
```

bude okienko s výsledkami prázdne. Jazyk SQL sa nebráni zložitejším konštrukciám, preto si teraz ukážeme, ako vybrať všetky štáty Severnej Ameriky a nechať si zobraziť iba názov štátu a hlavné mesto:

```
select name, capital from country where continent = „North America“
```

V tomto prípade sme dali databázovému stroju príkaz, aby vyhľadal všetky záznamy, v ktorých pole Continent obsahuje reťazec „North America“. Čo však v prípade, ak si chceme dať zobraziť všetky štáty, ktoré sa začínajú povedzme písmenkom „C“ ? Riešenie je veľmi jednoduché:

```
select * from country where name like „C%“
```

Rovnako jednoducho môžeme dosiahnuť, aby nám počítač vybral všetky krajiny, ktoré sa začínajú písmenkom „C“ a súčasne sa nachádzajú v Severnej Amerike:

```
select * from country where name like „C%“ and (continent = „North
America“)
```

Okrem znamienka = (t. j. rovná sa) môžeme používať aj znamienka <, >, <=, >=. Pomocou nasledujúceho SQL dopytu si necháme zobraziť všetky krajiny, v ktorých je počet obyvateľov vyšší ako sto miliónov:

```
select * from country where population > 100000000
```

Aby sme si to trošku skomplikovali, zúžime náš výber na krajiny, v ktorých je počet obyvateľov viac ako sto miliónov a ktoré sa súčasne nachádzajú v Severnej Amerike:

```
select * from country where population > 100000000 and continent
= „North America“
```

V tomto prípade bude výslednú množinu dát tvoriť iba jediný riadok, ktorý zobrazuje údaje o USA. Ako ste si už iste všimli, kľúčové slovo `AND` tu plní podobnú funkciu ako v Pascale. Počet kľúčových slov `AND` v SQL dopyte nie je (našťastie) obmedzený, preto si teraz necháme zobraziť zoznam severoamerických štátov, v ktorých je počet obyvateľov vyšší ako 50 000 a ich rozloha je väčšia ako 50 000 km^2:

```
select * from country where population > 50000 and continent = „North
America“ and area > 50000
```

Ďalšou skvelou vlastnosťou jazyka SQL je možnosť triediť údaje. Nasledujúci dopyt SQL nám zobrazí zoznam krajín zobrazených podľa abecedy:

```
select * from country order by name
```

Ekvivalentom tohto SQL dopytu je dopyt

```
select * from country order by name asc
```

pričom slovko „asc“ je skratkou slova ascending, čo znamená vzostupný. Údaje je však možné triediť aj zostupne:

```
select * from country order by name desc
```

Aby sme neostali pri jednoduchých príkazoch, napíšeme si opäť jeden zložitejší. Ten vyberie z databázy všetky severoamerické štáty, v ktorých je počet obyvateľov vyšší ako 50 000, ich rozloha je väčšia ako 50 000 km^2, a zoradí ich názvy podľa abecedy:

```
select * from country where population > 50000 and continent = „North
America“ and area > 50000 order by name
```

Zaujímavosťou je možnosť premenovať určité pole pomocou kľúčového slova `AS`. V nasledujúcom príklade si ukážeme, ako zmeniť názov poľa Capital na HlavneMesto:

```
select name, capital as HlavneMesto from country
```

Vo výslednej množine dát teda budeme vidieť dve polia: name a HlavneMesto. Pre úplnosť dodávam, že premenované pole nesmie obsahovať diakritiku ani medzery. Premenovávanie polí sa nám bude hodiť najmä vtedy, keď použijeme niektorú z agregátnych funkcií (aggregate functions), ako je napríklad funkcia SUM. Nasledujúci príklad nám spočíta plochu všetkých štátov Južnej Ameriky:

```
select sum(area) from country where continent = „South America“
```

Výsledkom tohto SQL dopytu bude jediný riadok s nepekným názvom „SUM OF area“. Pomocou kľúčového slova AS túto chybičku krásy veľmi rýchlo odstránime a pole nazveme PlochaCelkom:

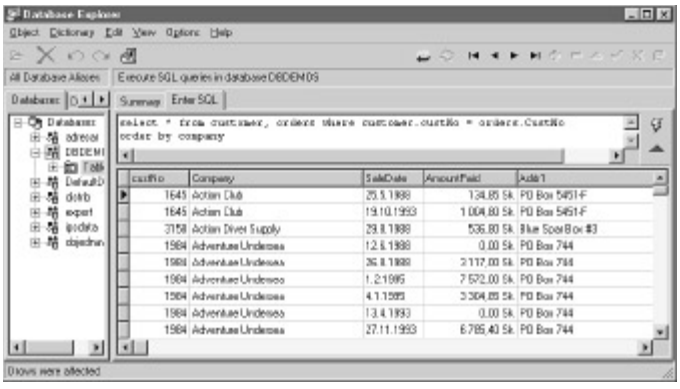
```
select sum(area) as PlochaCelkom from country where continent =
„South America“
```

Jazyk SQL umožňuje okrem výberu dát aj ďalšie základné operácie: vymazávanie, editovanie a pridávanie. Nasledujúci príkaz vymaže z databázy všetky štáty Južnej Ameriky:

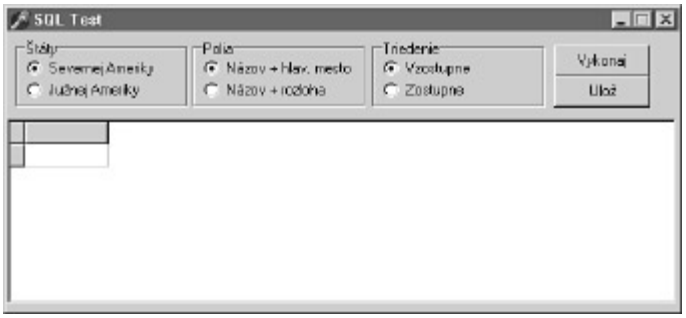
```
delete from country where continent=“South America“
```

Editovanie dát je rovnako jednoduché:

```
update country set continent=“Severna Amerika“ where continent=“North
America“
```



Obr. 1



Obr. 2

Nové záznamy sa do databázy pridávajú pomocou príkazu insert. Prvá zátvorka obsahuje zoznam polí, do ktorých budeme nové údaje zapisovať, a druhá zátvorka obsahuje hodnoty, ktoré sa do týchto polí majú zapísať. Nasledujúci príkaz pridá do databázy krajín Slovensko:

```
insert into country(name,capital,continent,area,population)
values („Slovensko“,“Bratislava“,“Europa“,“49036“,“5301000“)
```

Pokiaľ pracujete s lokálnymi databázami (dBase, Paradox), používajte na pridávanie nových záznamov metódy komponentu TTable. Pridávanie pomocou SQL dopytu je totiž znateľne pomalšie (to, samozrejme, neplatí pre SQL servery).

Ďalšou dôležitou črtou jazyka SQL je možnosť spájať dáta s niekoľkých databáz pomocou určitého kľúča. Možno si teraz poviete: Načo by sme ukladali svoje údaje do niekoľkých databáz, veď čím menej súborov, tým menej starostí. Nuž, to nie je vždy pravda. Zoberme si napríklad demo aplikáciu MastApp, ktorá je dodávaná s Delphi. Skladá sa z dvoch častí: údaje o zákazníkovi (súbor CUSTOMER.DB) a údaje o objednávkach (súbor ORDERS.DB). Pokiaľ by boli všetky záznamy v jednej databáze, nutne by dochádzalo k duplicite údajov. Databáza by totiž musela obsahovať nielen údaje o objednávkach, ale aj údaje o zákazníkoch. Keďže jeden zákazník môže dať niekoľko objednávok, museli by sa údaje o zákazníkovi opakovať pri každej objednávke. Keďže tieto údaje (t. j. údaje o zákazníkovi) sú nemenné, takýto návrh databázy nemožno nazvať inak ako plytvanie miestom: ak bude mať zákazník sto objednávok, údaje o ňom by museli byť uvedené stokrát, teda veľa každej objednávky. Ak by sa tieto údaje predsa len náhodou zmenili, museli by sme prejsť sto záznamov a každý upraviť. Preto je oveľa vhodnejšie rozdeliť databázu do dvoch súborov, pričom jeden bude obsahovať údaje o zákazníkovi a druhý údaje týkajúce sa jednotlivých objednávok. Základom databázy sú údaje o zákazníkovi, teda súbor CUSTOMER.DB. Detaily o jednotlivých objednávkach zákazníka nám prináša súbor ORDERS.DB. Takýto vzťah medzi súbormi sa po anglicky volá master – detail relationship.

SQL poskytuje prostriedky, ako takéto „master – detail“ tabuľky prepojiť. Spojením tabuliek vznikne množina dát, ktorá sa v angličtine označuje ako join. Spájanie tabuliek sa však musí uskutočniť na základe nejakého kľúča. V našom prípade je týmto kľúčom pole CustNo (skratka Customer Number, číslo zákazníka). Na základe tohto kľúča potom databázový stroj vytvorí zodpovedajúce páry záznamov. V praxi to vyzerá takto:

```
select Custno, Company, SaleDate from customer, orders where
customer.custNo = orders.CustNo order by company
```

Tento SQL dopyt spojí tabuľky CUSTOMER.DB a ORDERS.DB na základe kľúča, ktorým je pole CustNo. Výsledok potom zoradí podľa abecedy, pričom zobrazené budú iba polia CustNo, Company a SaleDate. Práve pole SaleDate je jasným dôkazom toho, že spojenie prebehlo správne. Aby sme toho videli viac, necháme si zobraziť všetky polia databázy:

```
select * from customer, orders where customer.custNo = orders.CustNo
order by company
```

Výsledok vidíme na obrázku číslo 1. Polia SaleDate a AmountPaid som zámerne umiestnil za pole Company, aby sme ľahšie vedeli rozlíšiť jednotlivé objednávky. Možno vás trošku mätie opakujúci sa názov firmy. Je to prirodzený jav: keďže došlo k spojeniu dvoch tabuliek, ku každej položke z tabuľky ORDERS.DB boli priradené údaje o zákazníkovi. Zákazník s viac ako jednou objednávkou bude teda uvedený viackrát (čo je koniec koncov logické).

SQL v Delphi

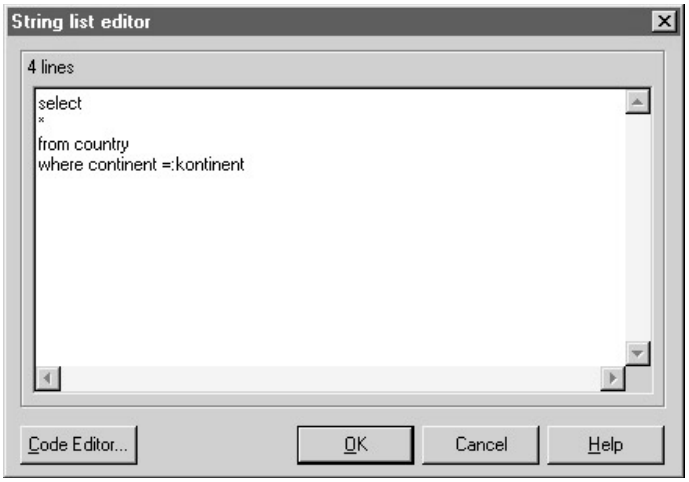
SQL môžeme používať aj v Delphi, a to vďaka komponentu TQuery. Práca s týmto komponentom je podobná práci s komponentom TTable. Hlavný rozdiel medzi týmito komponentmi je v tom, že výsledná množina dát je väčšinou „read – only“. Zámerne píšem „vo väčšine prípadov“, TQuery totiž za istých okolností vytvára aj editovateľné množiny dát, musia však byť splnené určité podmienky. „Read – only“ výsledok SQL dopytu je v dokumentácii Delphi označovaný ako „dead result set“. Pokiaľ SQL dopyt splnil podmienky „editovateľnosti“, je výsledná množina údajov označovaná ako „live result set“. Bližšie informácie nájdete v nápovedi pod heslom Live result sets.

Použitie komponentu je veľmi jednoduché: vyplníme vlastnosť SQL, vlastnosť DatabaseName nastavíme na DBDEMOS, pridáme komponent TDataSet a TDBGrid a vlastnosť Active nastavíme na true. Komponent TDBGrid nám zobrazí výsledok, a to dokonca bez nutnosti kompilovať program. Pre začiatok si môžete otestovať tých pár príkladov, o ktorých sme hovorili v predchádzajúcich odsekoch. Hlavným problémom takýchto „natvrdo“ napísaných SQL dopytov je to, že v praxi sa nevyužívajú často. Ako príklad si zoberme dopyt

```
select * from country where continent = „North America“
```

Čo ak zmeníme názor a budeme chcieť zoznam krajín Južnej Ameriky? Poviete si, že riešenie je predsa jednoduché, stačí pridať ďalší komponent TQuery, ktorý sa od toho predchádzajúceho líši iba tým, že vráti zoznam všetkých štátov Južnej Ameriky:

```
select * from country where continent = „South America“
```



Obr. 3

Toto riešenie je síce funkčné, je však až príliš pritiahnuté za vlasy. Čo keby sme mali databázu všetkých krajín a kontinentov sveta? Museli by sme ručne vytvoriť neuveriteľné množstvo komponentov TQuery, ktorých obsah by bol takmer totožný.

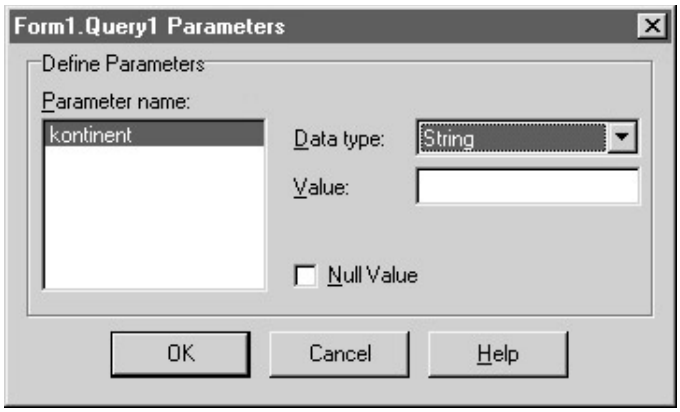
A čo v prípade, ak si chceme vybrať, ktoré polia sa nám majú zobraziť? V jednom prípade nám stačí názov krajiny a hlavné mesto, v druhom zasa názov krajiny a jej rozloha. Aj tento problém by sa dal vyriešiť uvedeným za vlasy pritiahnutým spôsobom, my ho však vyriešime oveľa elegantnejšie. Vysvetlíme si, ako odovzdávať SQL dopytom parametre a ukážeme si, ako „za jazdy“ modifikovať vlastnosť SQL komponentu TQuery. Náš miniprogram napíšeme tak, aby nám umožnil zvoliť si, či chceme vidieť štáty Severnej alebo Južnej Ameriky, ďalej nám umožní vybrať si, či chceme vidieť názov krajiny a hlavné mesto alebo názov a rozlohu, a napokon nám umožní zoradiť výslednú množinu údajov vzostupne alebo zostupne.

Dajme sa teda do práce. V prvom rade treba vytvoriť prítulné používateľské rozhranie, ktoré nám pomôže splniť našu úlohu. Moju predstavu o tom, ako takéto rozhranie vyzerá, vidíte na obrázku číslo 2. Potom napíšeme príslušný SQL dopyt (obr. 3). Teraz si určite položíte otázku, prečo som náš SQL dopyt napísal tak čudne, do niekoľkých riadkov. Odpoveďou vám budú nasledujúce odseky.

Vlastnosť SQL komponentu TQuery je vlastne objektom typu TStrings čiže objektom, ktorý obhospodaruje reťazce. Mohli by sme ju prirovnať k vlastnosti Lines komponentu TMemo, s ktorým sme sa oboznámili v druhej časti seriálu. K jednotlivým riadkom vlastnosti SQL možno pristupovať podobne ako k riadkom komponentu TMemo. Prečo to spomínam? Dôvod je jednoduchý: ak získame prístup k riadkom, budeme môcť „za jazdy“ (teda počas behu programu) modifikovať vlastnosť SQL. To sa nám zíde najmä v tom prípade, pokiaľ nedokážeme náš problém vyriešiť pomocou parametrov.

Mnohí z vás si teraz zrejme položia otázku, čo sú a na čo slúžia parametre. Odpoveď je jednoduchá: parametre SQL sú argumenty, ktoré budú odovzdané SQL dopytu predtým, ako sa vykoná. Inými slovami, parametre tu slúžia na ten istý účel ako v Pascale. Skôr než parametre použijeme, musíme najprv špecifikovať, akého dátového typu sú. Otvoríme preto vlastnosť Params. Ako vidíme, komponent TQuery si už rozanalyzoval náš SQL dopyt a zistil, že náš parameter sa volá kontinent. Aby sme mohli tento parameter používať, musíme správne nastaviť jeho dátový typ (obr. 4).

Vráťme sa však k vlastnosti SQL. Budeme ju (a teda aj SQL dopyt) meniť na základe toho, ktorá voľba v príslušnom komponente TRadioGroup je práve vyznačená. Najskôr však na formulár musíme pridať tlačidlo, ktoré nám SQL dopyt spustí. Prvý komponent TRadioGroup má za úlohu modifikovať vlastnosť SQL komponentu TQuery tak, aby



Obr. 4

výsledná množina údajov zahŕňala štáty buď Južnej, alebo Severnej Ameriky. Obslužná rutina OnClick nášho tlačidla Vykonaj bude vyzerať takto:

```
procedure TForm1.vykonajClick(Sender: TObject);
begin
 try
   query1.close;
   if staty.ItemIndex=0 then
   query1.ParamByName('kontinent').AsString:='North America'
   else query1.ParamByName('kontinent').AsString:='South America';
   query1.active:=true;
except
  raise;
end;//except
end;
```

Princíp je jednoduchý: pokiaľ je vybraná prvá voľba (t. j. štáty Severnej Ameriky), bude mať vlastnosť ItemIndex komponentu TRadioGroup hodnotu 0. V tom prípade nastavíme parameter kontinent na „North America“, v opačnom prípade na „South America“. Ako ste si už iste všimli, nastavovanie parametrov prebieha pomocou vlastnosti ParamByName, pričom argumentom v zátvorke je názov prametra SQL. Keďže tento parameter je typu string (pozri obr. č. 4), pomocou metódy AsString ho prekonvertujeme na požadovaný dátový typ. Posledný príkaz nám aktivuje samotný SQL dopyt.

Náš cieľ sme však ešte nedosiahli: ďalšou úlohou bude zohľadniť v SQL dopyte výber polí. V tomto prípade nám neostáva nič iné ako priamo modifikovať SQL dopyt, pričom modifikačná rutina bude vlastne obslužnou rutinou udalosti OnClick komponentu TRadioGroup:

```
procedure TForm1.poliaClick(Sender: TObject);
begin
     if polia.ItemIndex=0 then query1.sql[1]:='name, capital'
     else query1.sql[1]:='name, area';
end;
```

Celá vec však má jeden háčik: spustíme program a stlačíme tlačidlo Vykonaj. Na prvý pohľad by sa mohlo zdať, že program nefunguje. Spustíme ho preto ešte raz, zvolíme si štáty Južnej Ameriky a stlačíme tlačidlo Vykonaj. Bystrejší z vás akiste postrehli, že stojíme pred celkom triviálnym problémom: tento kúsok kódu (t. j. procedúru PoliaClick) je potrebné vykonať hneď po zobrazení formulára, inak bude program pracovať len vtedy, ak klikneme na komponent TRadioGroup a zmeníme voľbu z pôvodnej na inú. Inicializácia je jednoduchá:

```
procedure TForm1.FormShow(Sender: TObject);
begin
     PoliaClick(sender);
end;
```

Pri zobrazení formulára sa teda stane to isté, ako keby sme myšou vybrali jednu z volieb. Prezrite si procedúru PoliaClick a potom si ešte raz pozrite obrázok číslo 3. Riadky sa číslujú od nuly. Znamená to, že prvý riadok s príkazom select má číslo nula, druhý riadok (s hviezdičkou) má číslo jeden atď. Pokiaľ je vybraná prvá voľba, program musí zobraziť názov štátu a hlavné mesto. V tomto prípade je hviezdička nahradená reťazcom ‚name, capital‘. Výsledný SQL dopyt bude teda vyzerať takto:

```
select
name, capital
from country
where continent =:kontinent
```

Teraz už viete, prečo som SQL dopyt napísal tak čudne. V druhom prípade bude výsledný dopyt „zlepený“ do nasledujúce podoby:

```
select
name, area
from country
where continent =:kontinent
```

Pokiaľ si chcete moje tvrdenie overiť, nie je nič jednoduchšie: zaraďte bod prerušenia (breakpoint) na riadok `query1.active:=true`. Potom do okna Watches napíšte výraz *query1.sql[1]*. Program spustite a presvedčíte sa o správnosti mojej teórie. Implementácia triedenia pracuje na podobnom princípe ako výber polí:

```
procedure TForm1.triedenieClick(Sender: TObject);
begin
     if triedenie.ItemIndex=0 then query1.sql[4]:='order by name'
     else query1.sql[4]:='order by name desc';
end;
```

Urobme teraz malý test: spustime program a vyberme hociktorú formu triedenia. Vznikne výnimka s chybovým hlásením „List index out of bounds“. Viete, čo sa stalo? Prezrite si ešte raz obrázok číslo 3. Všimnite si, že v hornej časti okna je text „4 lines“. Pokúšali sme sa nastaviť obsah štvrtého riadka, ktorý vlastne ani neexistuje (nezabudnite, že riadky sa číslujú od nuly, máme teda štyri riadky s číslami 0, 1, 2 a 3). Preto otvoríme vlastnosť SQL a pridáme jeden prázdny. Nesmieme zabudnúť ani na inicializáciu:

```
procedure TForm1.FormShow(Sender: TObject);
begin
    PoliaClick(sender);
    TriedenieClick(sender);
end;
```

Na pomalších strojoch môže trvať vykonanie SQL dopytu aj niekoľko sekúnd. Počas týchto niekoľkých sekúnd bude program „hluchý“. Ako však dáme používateľovi na vedomie, že program pracuje, a teda je zbytočné klikať myšou hore-dole? Riešenie je veľmi jednoduché: počas vykonávania SQL dopytu zmeníme kurzor myši na presýpacie hodiny:

```
procedure TForm1.vykonajClick(Sender: TObject);
begin
 try
  try
   screen.Cursor:=crHourglass;
   query1.close;
   if staty.ItemIndex=0 then
   query1.ParamByName('kontinent').AsString:='North America'
   else query1.ParamByName('kontinent').AsString:='South America';
   query1.active:=true;
 except
  raise;
 end;//except
 finally
  screen.Cursor:=crDefault;
 end;//finally
end;
```

Všimnite si použitie bloku `try...finally`. Pokiaľ by sme ho vynechali a počas vykonávania SQL dopytu by nastala výnimka, kurzor by si ponechal tvar presýpacích hodín a používateľ by si myslel, že program pracuje, aj keď v skutočnosti už nepracuje (keďže pri vzniku výnimky sa procedúra, v ktorej výnimka vznikla, ukončí). Tento blok nám zabezpečí, že aj v prípade vzniku výnimky sa bude kurzor myši správať korektne.

V praxi sa relatívne často stretávame s problémom ukladania dát vygenerovaných SQL dopytom. Na tento účel nám dokonale poslúži metóda BatchMove komponentu TTable. Najprv teda musíme vytvoriť nejakú tabuľku. Bystrejší z vás sa teraz možno pýtajú, ako máme vytvoriť tabuľku, keď nepoznáme počet, názvy ani typy polí, ktoré sa budú nachádzať v množine dát vygenerovanej SQL dopytom. Odpoveď je triviálna: o štruktúru dát sa netreba starať. Metóda BatchMove si tabuľku automaticky prispôsobí. Nám teda stačí vytvoriť tabuľku s jediným poľom, na ktorého názve a type nezáleží (pretože bude potom aj tak nahradené iným poľom). Samozrejme, nesmieme zabudnúť pridať komponent TTable a ďalšie tlačidlo, ktoré nazveme Uloz. Samotný kód je veľmi jednoduchý:

```
procedure TForm1.ulozClick(Sender: TObject);
begin
    Table1.BatchMove(query1,batCopy);
end;
```

Prvým parametrom je objekt triedy TBDEDataSet. Je to vlastne zdroj údajov, ktorým je v našom prípade komponent TQuery (namiesto ktorého by sme mohli použiť komponent TTable). Druhým parametrom je spôsob nakladania s údajmi pri operácii

presunu, podrobnosti nájdete v nápovedi Delphi. Náš krátky výlet do sveta SQL sa týmto záverečným kúskom kódu končí. Je potrebné uviesť, že problematika SQL je oveľa obsiahlejšia a jej kompletný rozbor by stačil na jeden samostatný seriál, preto pokladajte tento článok iba za veľmi stručný úvod do tejto problematiky.

Nabudúce

Doteraz sme pracovali s databázovými súbormi Paradoxu. Pristupovali sme k nim prostredníctvom komponentov TTable a TQuery. V budúcej časti seriálu si ukážeme, ako pristupovať k „nedatabázovým" súborom. Povieme si niečo o procedúrach Read, ReadLn, Write, WriteLn, BlockRead, BlockWrite a napíšeme si niekoľko malých programov: utilitu na export databázy do textového formátu, ďalej program, ktorý nám umožní vybrať si EXE súbor a následne vypíše, či ide o 16-bitový (NE) alebo 32-bitový (PE) EXE súbor, a napokon si ukážeme jednoduchú rutinu na kopírovanie súborov. Nadnes je to všetko, dovidenia nabudúce.

Problémy s ActiveX

V predchádzajúcej časti seriálu som sa zmienil o problematike registrácie ActiveX komponentov. V tomto krátkom článku si ukážeme, ako vyzerá riešenie tohto problému v praxi. Aby som bol konkrétnejší, ukážeme si postup, ako zistiť, či určitá DLL knižnica používa ešte ďalšiu DLL knižnicu. Okrem toho si povieme, ako zaregistrovať ActiveX komponent. Ako „pokusný králik" nám bude slúžiť program Web Browser, ktorý je dodávaný s Delphi ako demo aplikácia (nachádza sa v adresári Delphi X\Demos\Coolstuff). Skôr než sa pustíme do práce, chcel by som vážených čitateľov upozorniť na to, že tento článok nie je určený pre začiatočníkov.

Pripomínam, že základný problém spočíva v tom, že súbory OCX nestačí iba jednoducho prekopírovať do systémového adresára Windows; ActiveX komponenty treba zaregistrovať. V dokumentácii Delphi nájdete odkaz na funkcie DllRegisterServer a DllUnregisterServer. Bohužiaľ, ani po niekoľkých dňoch intenzívneho experimentovania sa mi ich nepodarilo spojazdniť, a tak som sa rozhodol ísť na problém z opačného konca. OCX súbory sú v podstate DLL súbory, ktoré exportujú dve základné funkcie: DllRegisterServer a DllUnregisterServer. To znamená, že tieto dve funkcie je možné naimportovať tak isto, ako by sme importovali funkcie z klasickej DLL knižnice:

```
procedure RegServer;external 'html.ocx' name 'DllRegisterServer';
procedure UnregServer; external 'html.ocx' name 'DllUnregisterServer';
```

Názvy importovaných funkcií, ktoré sú uvedené za kľúčovým slovom name, sú „case-sensitive", takže ich odpisujte pozorne.

Možno si teraz položíte otázku, ako som zistil, že komponent THTML je uložený v súbore HTML.OCX. Nuž, je to veľmi jednoduché: z menu Component stačí vybrať položku Import ActiveX Control a zobrazí sa nám zoznam všetkých registrovaných ActiveX komponentov. Nie je problém vyhľadať si v tomto zozname potrebný komponent, v našom prípade ide o NetManage HTML Client Control verzie 1.0. Pod zoznamom komponentov sa nachádza názov OCX súboru, v ktorom je komponent uložený.

Bohužiaľ, zoznam potrebných súborov ešte ani zďaleka nie je kompletný, pretože súbor HTML.OCX používa ešte ďalšie DLL knižnice. Zistil som to, keď som (po drobnej úprave zdrojového kódu) prekopíroval Web Browser na iný počítač. Našťastie sú to iba obyčajné DLL súbory (nie ActiveX komponenty), ktoré stačí jednoducho prekopírovať. Problém je v tom, že v dokumentácii som nikde nenašiel informácie o tom, ktoré súbory HTML.OCX ešte používa. Neostávalo mi teda nič iné ako použiť metódu pokus – omyl (postupne som na druhý počítač kopíroval súbory, ktoré si Web Browser žiadal).

Našťastie neskôr mi napadlo aj elegantnejšie riešenie: keďže súbory OCX sú vlastne DLL, je možné zistiť si zoznam importovaných a exportovaných funkcií. Na to nám poslúži utilita Turbo Dump (TDUMP.EXE), ktorá sa nachádza v adresári Delphi X\Bin. Jej použitie je jednoduché:

```
TDUMP HTML.OCX -e HTM.DMP
```

Voľba –E znamená „Force executable file display", čiže nám zobrazí základné údaje o vykonateľnom súbore (PE hlavičku, údaje o jednotlivých sekciách, zoznam importovaných a exportovaných funkcií atď.). Prv než nazrieme do súboru HTM.DMP, ukážeme si, ako upraviť spúšťací kód programu Web Browser:

```
program Webbrows;

uses
  Forms,SysUtils,
  main in 'Main.pas' {MainForm},
  DocSrc in 'Docsrc.pas' {DocSourceFrm},
  About in 'About.pas' {AboutBox};

{$R *.RES}

procedure RegServer;external 'html.ocx' name 'DllRegisterServer';
procedure UnregServer; external 'html.ocx' name 'DllUnregisterServer';

begin
  RegServer;
  if AnsiLowerCase(paramstr(1))='/unreg' then
  begin
     UnregServer;
     exit;
  end;

  Application.CreateForm(TMainForm, MainForm);
  Application.CreateForm(TDocSourceFrm, DocSourceFrm);
  Application.Run;
end.
```

Program som prerobil tak, aby hneď po spustení automaticky zaregistroval súbor HTML.OCX. Pre prípad, že budem chcieť Web Browser neskôr vymazať, rozhodol som sa pridať aj možnosť deregistrácie. Pokiaľ program spustíte s parametrom /unreg, ActiveX komponent bude automaticky deregistrovaný a ukončený. Pripomínam, že deregistráciou sa komponent (teda OCX súbor) nevymaže.

Vráťme sa však späť k súboru HTM.DMP. Po krátkej prehliadke zistíme, že súbor HTML.OCX importuje funkcie zo súborov MSVCRT40.dll, NMOCOD.DLL, NMW3VWN.DLL, OLEPRO32.DLL, KERNEL32.dll, USER32.dll, ole32.dll, ADVAPI32.dll, OLEAUT32.dll, GDI32.dll. Keďže väčšina týchto súborov je súčasťou Windows, nemusíme sa obávať, či budú na cieľovom počítači k dispozícii. Nás budú zaujímať predovšetkým súbory NMOCOD.DLL a NMW3VWN.DLL. Bohužiaľ, ani po prekopírovaní týchto súborov program nebude fungovať. Dôvod je jednoduchý: knižnica NMOCOD.DLL tiež importuje funkcie z inej DLL. Našou úlohou je zistiť, z ktorej:

TDUMP NMOCOD.DLL –e NMOCOD.DMP

Po podrobnejšom preskúmaní súboru zistíme, že okrem štandardných DLL knižníc táto knižnica používa súbor NMSCKN.DLL. Tento súbor však tiež môže importovať funkcie z iných DLL knižníc, preto ho musíme preskúmať:

```
TDUMP NMSCKN.DLL –e NMSCKN.DMP
```

Našťastie tento súbor používa iba štandardné DLL knižnice. Po prekopírovaní tohto súboru na druhý počítač Web Browser síce fungoval, no pýtal si ďalšiu knižnicu – NMORENU.DLL. Príčinou môže byť to, že sme nepreskúmali súbor NMW3VWN.DLL, ktorý možno túto knižnicu používa:

```
TDUMP NMW3VWN.DLL –e NMW3VWN.DMP
```

Po krátkej prehliadke zistíme, že táto knižnica (teda NMW3VWN.DLL) importuje funkcie iba zo štandardných DLL knižníc. Súbor NMORENU.DLL je však aj napriek tomu potrebné prekopírovať na cieľový počítač. Pre istotu som ho preskúmal, importoval však iba funkcie z KERNEL32.DLL. Napokon nám ostáva už iba skopírovať AVI súbor, ktorý Web Browser používa.

Na záver by som chcel poznamenať, že tento článok v žiadnom prípade nie je všeliekom na „registračné" problémy s ActiveX. Pracoval som s verziou Delphi 3 a ActiveX komponentmi, ktoré sú s ňou dodávané. Je možné, že vo vyšších verziách sa budú v článku spomínané DLL knižnice (samozrejme okrem systémových) volať úplne inak. Preto berte tento príspevok iba ako akéhosi sprievodcu, ktorý vám ukáže správnu cestu, nie však konečné riešenie.

Desiata časť: Práca so súbormi

V tejto časti si povieme niečo o súborovej podpore jazyka Pascal. Konkrétne budeme hovoriť o textových, binárnych, typových a netypových súboroch. Nadobudnuté vedomosti potom zúročíme v niekoľkých zaujímavých programoch.

Základné pojmy

Na úvod si položme otázku: Čo je vlastne súbor? Súbor je organizovaná skupina dát, ktoré sú uložené na diskete, hard disku alebo inom médiu. Organizácia dát v súboroch je značne závislá od toho, na aký účel sa súbory používajú, resp. na aký účel sa používajú dáta obsiahnuté v súboroch. Doteraz sme pracovali s databázovými súbormi Paradoxu; o organizácii dát v takýchto typoch súborov teda už vieme pomerne veľa. Sebakriticky však musíme dodať, že o fyzickej štruktúre týchto súborov nevieme nič. Je zbytočné tvrdiť, že znalosť fyzickej štruktúry súborov by nám bola na niečo dobrá; databázové komponenty Delphi sa totiž dôsledne starajú o to, aby sme o vnútornom usporiadaní „paradoxovských" súborov nemuseli nič vedieť. Podobne je to s grafickými komponentmi, ktoré slúžia na zobrazenie obrázkov. O fyzickej štruktúre a ani o obrazovom formáte nemusíme vôbec nič vedieť, o správnu interpretáciu

súboru sa postará príslušný komponent. Keď však začnete písať pokročilejšie aplikácie, nepochybne sa dopracujete k tomu, že budete musieť siahnuť po príkazoch na manipuláciu so súbormi. Našťastie pre Delphi to nie je problém. Príkazy na manipuláciu so súbormi sú takmer identické so „súborovým repertoárom“ Turbo Pascalu. Pokiaľ ste teda zvládli prácu so súbormi v Turbo Pascale, nebude pre vás problém pochopiť princípy, na ktorých sa zakladá súborová podpora Delphi.

Súbory by sme mohli rozdeliť do dvoch skupín: textové a binárne. Príkladom textového súboru môžu byť napríklad súbory programu Notepad alebo INI súbory. Hoci som použil slovo „textový“, vonkoncom nejde len o súbory s príponou TXT. Základnou črtou textových súborov je to, že používajú znaky, ktoré sa bežne vyskytujú v textových dokumentoch: všetky písmená abecedy, výkričník, bodka, čiarka, percento, hviezdička atď. Druhou skupinou sú binárne súbory. Sem by sme mohli zaradiť súbory s koncovkou EXE, DLL, SYS a všetky grafické formáty. Binárne súbory sa delia na typové a netypové. Rozdiel medzi nimi je najmä v deklarácii, napr. typový súbor, ktorý sa skladá iba z údajov typu integer, deklarujeme ako:

```
Var
  Subor: file of integer;
```

Deklarácia netypového súboru bude vyzerať takto:

```
Var
    Subor:file;
```

Podporu binárnych súborov využijeme napríklad pri rutine, ktorá bude kopírovať súbory do určitého adresára. Vtedy je vždy potrebné zdrojový a cieľový súbor deklarovať ako binárny a netypový. Možno sa niektorí z vás teraz pýtajú, prečo tu filozofujem o kadejakých typových a netypových súboroch, keď kopírovanie súborov sa dá doriešiť relatívne jednoduchou rutinou, ktorá bude využívať príkazy ReadLn a WriteLn. Keď som sa kedysi pred mnohými rokmi zaoberal QuickBasicom, použil som podobný prístup. Čakalo ma však nemilé prekvapenie: program neprekopíroval správne súbory typu EXE a COM (samozrejme, neprekopíroval by ani iné binárne súbory). Príčina bola jednoduchá: príkazy, ktoré som použil, boli určené na prácu s textovými, a nie binárnymi súbormi. Po podrobnejšej analýze som zistil, že program pracuje správne, až kým nenarazí na kombináciu znakov Carriage Return a Line Feed, ktoré sa nachádzajú na konci každého riadka textového súboru. Tieto znaky totiž v textových súboroch znamenajú prechod na nový riadok. Binárne súbory však nie sú rozdelené do riadkov. Riadky boli zavedené len preto, aby človeku umožnili lepšiu orientáciu v texte. Binárne súbory čítajú len programy, ktoré im dokonale rozumejú, a nepotrebujú ich deliť na riadky. Môj basicový program v EXE súbore narazil na uvedenú kombináciu znakov a „myslel si“, že má prejsť na nový riadok (hoci mal pokračovať tam, kde prestal). Aby som to zhrnul: rutiny ako ReadLn a WriteLn nie sú vhodnými kandidátmi na prácu s binárnymi súbormi. Opačne to však neplatí, pretože procedúry BlockRead a BlockWrite, ktoré slúžia na prácu s binárnymi súbormi, môžeme bez obáv použiť aj pri práci s textovými súbormi. V praxi to však pričasto robiť nebudeme, pretože prístup k textovým súborom pomocou ReadLn a WriteLn je oveľa jednoduchší.

Jednoduchý export dát

Ako ukážkový príklad práce s textovými súbormi nám bude slúžiť program, ktorý dokáže akúkoľvek tabuľku Paradox skonvertovať do textového formátu. Tento príklad síce priveľký zmysel nedáva, naučíte sa však na ňom tie najnevyhnutnejšie základy a okrem toho aj niečo nové z oblasti databáz. Vzhľad používateľského rozhrania našej aplikácie nie je dôležitý, preto vám ukážem iba základnú procedúru, ktorá export dát vykonáva:

```
procedure TForm1.Db2Txt;
var
  DBPath, TxtFilePath:string;
  NazovTabulky,TxtFileName:ShortString;
  i:integer;
  ExportFile:textfile;
  Polia,Bodka:integer;
  TxtBuffer, NazovPola:string;
begin
  try
    try
      if not OpenDlg.Execute then exit;
      if OpenDlg.FileName='' then exit;
      DBPath:=ExtractFileDir(OpenDlg.FileName);
      NazovTabulky:=ExtractFileName(OpenDlg.FileName);
      TxtFilePath:=DBPath;
      Bodka:=Pos('.',NazovTabulky);
       TxtFileName:= TxtFilePath + '\' + Copy(NazovTabulky,1,Bodka-1)+
'.txt';

      Screen.Cursor:=crHourGlass;
      with Table do
      begin
        DatabaseName:=DBPath;
        TableName:=NazovTabulky;
        Open;
        First;
      end;//with

      AssignFile(ExportFile, TxtFileName);
      Rewrite(ExportFile);
        for i :=0  to Table.RecordCount do
        begin
          for Polia :=0  to Table.FieldCount-1 do
          begin
            NazovPola:=Table.Fields[polia].FieldName;

            if not Table.Fields[polia].IsBlob then
                                  TxtBuffer:=NazovPola  +  ':  ' +
Table.Fields[polia].AsString
             else
              TxtBuffer:=NazovPola + ': BLOB';
              Writeln(ExportFile,TxtBuffer);
            end;
          Writeln(ExportFile,'**********************************');
          Table.Next;
          end; //for i :=0  to Table.RecordCount do
          CloseFile(ExportFile);
    except
      raise;
    end;//except
    finally
      Screen.Cursor:=crDefault;
      Table.Close;
    end;//finally
end;
```

Hoci sa procedúra môže zdať na prvý pohľad zložitá, pracuje na veľmi jednoduchom princípe. V prvom rade je potrebné vybrať databázový súbor, z ktorého budeme čerpať údaje. Súbor TXT, do ktorého sa textové údaje budú ukladať, bude mať rovnaký názov a bude sa nachádzať v tom istom adresári ako DB súbor, líšiť sa bude iba príponou. Premenná Bodka uchováva pozíciu bodky v reťazci, ktorý obsahuje názov súboru. Keďže my potrebujeme iba názov súboru bez prípony, musíme z pôvodného reťazca vybrať Bodka-1 znakov:

```
Copy(NazovTabulky,1,Bodka-1)
```

Aby sa textový súbor nachádzal v tom istom adresári ako pôvodný databázový súbor, musíme k jeho názvu pridať aj cestu:

```
TxtFileName:=  TxtFilePath  +  '\'  +  Copy(NazovTabulky,1,Bodka-1)  +
'.txt';
```

Keďže nepracujeme s aliasmi (bolo by zrejme zbytočné vytvárať alias pre jednorazovú operáciu), vlastnosť DatabaseName bude obsahovať adresár, v ktorom sa tabuľka nachádza:

```
DatabaseName:=DBPath;
```

Databázu potom otvoríme a pomocou metódy First sa presunieme na prvý záznam. Teraz prichádzajú na rad samotné I/O (Input/Output) rutiny: pomocou AssignFile najprv priradíme názov súboru súborovej premennej. Súbor potom otvoríme pomocou procedúry Rewrite. Je potrebné poznamenať, že pokiaľ otvárate súbor touto procedúrou, všetky predchádzajúce údaje budú zo súboru vymazané. Inými slovami, stane sa to isté, ako keby ste súbor najprv vymazali a potom znova vytvorili. Teraz nasledujú cykly, ktoré na prvý pohľad vyzerajú dosť mätúco. Prvý je cyklus:

```
for i :=0  to Table.RecordCount do....
```

Tento cyklus je potrebný preto, aby boli postupne spracované všetky záznamy tabuľky. Ich počet zistíme pomocou funkcie RecordCount. Všimnite si metódu Next na konci tohto cyklu: pomocou nej sa dostaneme na ďalší záznam. Pri získavaní údajov o názvoch polí je potrebný ďalší cyklus:

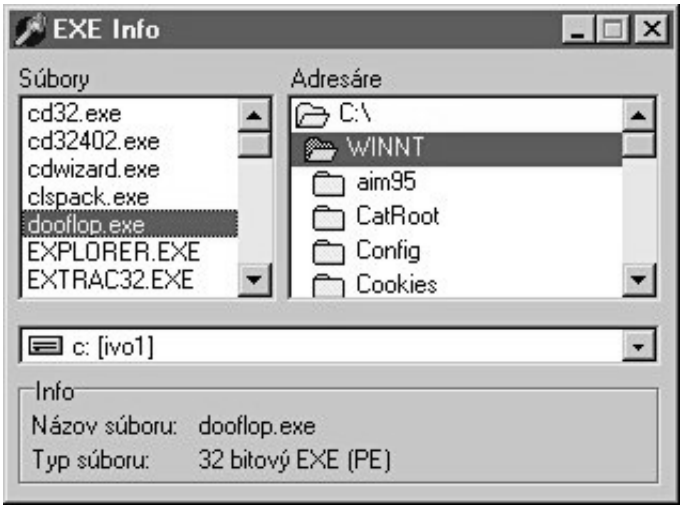
```
for Polia :=0  to Table.FieldCount-1 do...
```

Musíme prejsť postupne všetky polia tabuľky a pomocou metódy FieldName zistiť ich meno. Možno sa pýtate, prečo som cyklus nezapísal radšej takto:

```
for Polia :=1  to Table.FieldCount do
```

Dôvod je veľmi jednoduchý: číslovanie polí sa začína od nuly. Pokiaľ by som cyklus zapísal takto, musel by som jednotku odčítať pri odovzdávaní parametra metóde Fields:

```
NazovPola:=Table.Fields[polia-1].FieldName;
```



Obr. 1

Zaujímavosťou je použitie metódy IsBlob. Akiste si spomínate, že v našom adresári osôb máme aj pole typu BLOB, do ktorého ukladáme obrázky. Tie sa, bohužiaľ, v textovom súbore vidieť nedajú, preto v prípade BLOB poľa ponecháme záznam prázdny (t. j. vo výstupnom súbore bude iba názov poľa, za ktorým bude nasledovať slovo „BLOB“). Samotný text sa do súboru zapíše pomocou procedúry WriteLn. Prvým parametrom je premenná typu súbor a druhým je reťazec, ktorý chceme do súboru zapísať. Aby bol výstupný súbor prehľadnejší, sú jednotlivé záznamy oddelené hviezdičkami. Po vyexportovaní všetkých záznamov sa súbor zatvorí pomocou procedúry CloseFile. Vidíte, ani to nebolelo. Naučili ste sa niečo nové nielen z oblasti súborov, ale aj databáz.

Utilita EXE Info

Druhým ukážkovým príkladom bude malá utilitka, ktorá dokáže rozlíšiť 16- a 32-bitové EXE súbory. Netreba pripomínať, že tentoraz budeme pracovať nie s textovými, ale binárnymi netypovými súbormi. Skôr než sa pustíme do tvorby programu, opíšem vám princíp, na ktorom pracuje. Ako všetci vieme, existujú dva typy vykonateľných súborov pre Windows: 16-bitové a 32-bitové. Operačné systémy Windows 95 a NT musia pri spúšťaní EXE súboru zistiť, o aký súbor ide (t. j. či 16- alebo 32-bitový). Tieto (a mnohé iné) údaje sa dajú vyčítať z hlavičky EXE súboru. Pre hlavičku 32-bitového súboru je príznačné, že sa v nej nachádza slovíčko PE (skratka Portable Executable), za ktorým nasledujú dva znaky NULL (t. j. nulté znaky ASCII tabuľky). V 16-bitových aplikáciách zasa nájdeme slovíčko NE. Jediné, čo musíme zistiť, je to, ktoré z týchto dvoch slovíčok obsahuje hlavička nami skúmaného EXE súbor. Nie je to však až také jednoduché, ako to na prvý pohľad vyzerá. Poloha tohto „zázračného“ slova je iná pri každom EXE súbore. Našťastie existuje spôsob, ako ju spoľahlivo zistiť. V dokumentácii PE formátu som našiel nasledujúci návod: na offsete 60 sa nachádza 32-bitová hodnota, ktorá nám polohu tohto „zázračného“ slovíčka určí. Na moje prekvapenie ten istý spôsob fungoval aj pri 16-bitových EXE súboroch. Tí pokročilejší z vás (ktorí týmto údajom rozumejú) môžu nasledujúci odsek preskočiť, začiatočníci nech pozorne čítajú.

Pojem offset by sa v tomto prípade dal veľmi voľne preložiť ako posun. V našom prípade sa potrebujeme posunúť o 60 znakov od začiatku súboru. Na tomto mieste potom nájdeme 32-bitovú hodnotu, ktorá nám určí offset nášho „zázračného slova“. Pri riešení podobných problémov je nesmierne dôležité vedieť, koľko bitov zaberá nami hľadaný údaj; na základe tejto informácie totiž stanovíme typ premennej, do ktorej údaj načítame. V našom prípade to môže byť buď integer, alebo doubleword čiže dword. Obidva typy premenných totiž zaberajú 32 bitov.

Dajme sa teda do práce: vytvoríme novú aplikáciu a pridáme komponenty TFileListBox, ktorý nazveme Files, TDirectoryListBox, ktorý nazveme Dir, TDriveComboBox, ktorý nazveme Disk, komponent TGroupBox a dva komponenty TLabel (prvý nazveme NazovSubor a druhý TypSuboru). Aby komponent TDirectoryListBox „vedel“, z ktorej diskovej jednotky má čítať adresáre, musíme ho prepojiť s príslušným komponentom TDriveComboBox. Vlastnosť DirList komponentu TDriveComboBox nastavíme na Dir. Keďže komponent TFileListBox má zobrazovať súbory z adresára, ktorý je práve otvorený v TDirectoryListBoxe, musíme aj tieto dva komponenty prepojiť: vlastnosť FileList komponentu TDirectoryListBox nastavíme na Files. Takto poprepájané komponenty už budú spolupracovať správne. Musíme však ešte nastaviť filter tak, aby nám zoznam súborov ukazoval iba EXE súbory. Vlastnosť Mask komponentu TFileListBox nastavíme na *.exe.

Samotné zisťovanie informácií o EXE súbore bude prebiehať tak, že používateľ dvakrát klikne na vyznačený súbor. Príslušná obslužná rutina bude vyzerať takto:

```
procedure TForm1.FilesDblClick(Sender: TObject);
var subor:file;
buf:array[1..2] of char;
Signature:string;
SignaturePosition:dword;
bytesread:integer;
begin
 try
    NazovSuboru.Caption:=ExtractfileName(files.FileName);
    FileMode:=0;
    AssignFile(subor,files.FileName);
    Reset(subor,1);
  try
    Seek(subor,60);

BlockRead(subor,SignaturePosition,SizeOf(SignaturePosition),bytesread);
    Seek(subor,SignaturePosition);
    BlockRead(subor,buf,SizeOf(buf),bytesread);
    Signature:=buf;
    TypSuboru.Caption:='Neznámy';
    if Signature='PE' then TypSuboru.Caption:='32 bitový EXE (PE)';
    if Signature='NE' then TypSuboru.Caption:='16 bitový EXE (NE)';
  finally
    CloseFile(subor);
  end;//finally
 except
  raise;
 end;//except
end;
```

Rozoberme si teraz spôsob práce tejto rutiny. V prvom rade treba zistiť názov EXE súboru, s ktorým budeme pracovať. Aby hlavné okno programu nezaberalo priveľa miesta, budeme zobrazovať iba názov súboru (bez cesty). Nastavenie premennej FileMode na hodnotu nula nám zabezpečí, že súbory budú otvorené iba na čítanie. Pokiaľ by sme tento riadok vynechali, nepodarilo by sa nám otvoriť EXE súbor nijakého bežiaceho programu. Implicitná hodnota tejto premennej je 2, čo v praxi znamená otvorenie súboru na čítanie aj zápis. Je jasné, že nie je možné otvoriť na zápis už bežiaci EXE súbor. Samotný súbor otvoríme pomocou procedúry Reset. Prvým parametrom je premenná typu file, inými slovami, súbor, ktorý ideme otvoriť. Druhým parametrom je veľkosť záznamu, v dokumentácii Delphi označovaná ako „record size“. My sa budeme pohybovať po jednotlivých bajtoch, preto tento parameter nastavíme na 1 (pokiaľ by sme tento parameter vynechali, vedeli by sme údaje načítavať iba v 128-bajtových blokoch, čo je implicitná hodnota). Na offset 60 sa presunieme pomocou procedúry Seek. Potom nasleduje samotné načítanie čísla, ktoré nám udáva polohu „zázračného slova“. Čítanie sa realizuje prostredníctvom procedúry BlockRead. Prvým parametrom je premenná typu súbor (čiže súbor, z ktorého sa majú údaje načítať). Druhým parametrom je bufer, t. j. miesto v pamäti, do ktorého sa majú údaje načítať. Množstvo údajov, ktoré treba načítať, nám udáva tretí parameter. Podotýkam, že pri určovaní veľkosti bloku dát, ktorý sa má načítať, je dobrým zvykom používať funkciu SizeOf; predídete tak mnohým nepríjemným prekvapeniam. Posledný parameter nevypĺňate vy, ale procedúra. Pomocou neho možno zistiť, aké množstvo údajov bolo v skutočnosti načítané. Vhodný je najmä pri kontrole, keď zisťujete, či ste naozaj prečítali toľko bajtov, koľko ste plánovali. Druhý príkaz Seek nás presunie na miesto, kde sa nachádza nami hľadaná hodnota (čiže slovko PE alebo NE). Potom nasleduje jej načítanie. Keďže polia typu char a reťazce typu string sú navzájom kompatibilné, môžeme obsah bufera buf priradiť premennej Signature.

Komponent TLabel, ktorý vypisuje typ programu, bude implicitne obsahovať text „Neznámy“. Tento text sa vypíše napríklad vtedy, pokiaľ natrafíte na nejaký MS DOS program. Pokiaľ bude hlavička EXE súboru obsahovať slová PE alebo NE, vypíše sa príslušný identifikačný text. Konečnú podobu programu vidíme na obrázku č. 1.

Všimnite si usporiadanie blokov `try...finally` a `try...except`, ktoré sa vám na prvý pohľad možno bude zdať nezvyčajné, má však opodstatnenie. Urobme takýto pokus: riadok s príkazom AssignFile prepíšme tak, aby obsahoval názov neexistujúceho súboru. Pri pokuse o otvorenie takéhoto súboru vznikne výnimka a program „skočí“ na príkaz raise. Je zrejmé, že v tomto prípade by nestačil iba klasický `try...finally` blok, pretože potom by sa program pokúsil zatvoriť súbor, ktorý ešte ani nebol otvorený. Z toho dôvodu je blok `try...finally` umiestnený až za príkaz Reset. Ak sa vyskytne výnimka v bloku, ktorý číta dáta zo súboru, začne sa vykonávať blok `finally...end` a súbor bude uzavretý. Opäť si to môžeme overiť na jednoduchom pokuse: do procedúry pridáme dve premenné typu integer, pričom jedna z nich bude mať hodnotu nula. Potom ich umiestnime za procedúru Seek a vydelíme ich. Vznikne výnimka „Division by zero“, v dôsledku čoho sa vykoná blok `finally...end` a súbor za zatvorí.

Ako to už býva zvykom, aj táto naša aplikácia má ešte jednu dosť závažnú chybičku krásy: vyberte jednotku A:, disketu však nevložte. Po chvíľke sa objaví dialógové okno, ktoré vám ponúkne možnosť zrušiť alebo zopakovať neúspešný pokus o čítanie (Abort, Retry, Ingore). V prípade, že kliknete na Abort alebo Ignore, dostanete nepekné chybové hlásenie „I/O Error 21“. Našťastie tento problém sa dá veľmi jednoducho odstrániť. V prvom rade je potrebné vymazať obsah vlastnosti DirList komponentu TDriveComboBox. Túto vlastnosť budeme nastavovať ručne, ale až po tom, čo si overíme,

Kompletný výpis jednotky FCopy:

```
unit fcopy;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs,
  StdCtrls, ComCtrls, FileCtrl;
type
  TCopyForm = class(TForm)
    ProgressBar: TProgressBar;
    label1: TLabel;
    Label2: TLabel;
    zrus: TButton;
    SourceFiles: TListBox;
    odkial: TLabel;
    kam: TLabel;
    procedure StartCopy;
    procedure zrusClick(Sender: TObject);
  private
    { Private declarations }
  public
    TargetDir:string;
    { Public declarations }
  end;
var
  CopyForm: TCopyForm;
implementation
var
ZatialPrekopirovane:TStringList;
AllFilesClosed,Zastavene:boolean;
Subor, NovySubor:file;
{$R *.DFM}
procedure CloseAllFiles;
begin
  if AllFilesClosed then exit;
  CloseFile(subor);
  CloseFile(NovySubor);
end;
procedure TCopyForm.StartCopy;
var VelkostSpolu:integer;
NacitaneCelkom,i:integer;
BytesRead,BytesWritten:integer;
NewFileName:string;
Buffer:array[1..250000] of byte;
PovodnyDatum:integer;
handle:THandle;
begin
try
  ZatialPrekopirovane:=TStringList.Create;
  Zastavene:=false;
  AllFilesClosed:=false;
  FileMode:=0;
  VelkostSpolu:=0;
  NacitaneCelkom:=0;
  ProgressBar.Position:=0;
  CopyForm.odkial.Caption:='Zistujem veľkost súborov...';
  CopyForm.kam.Caption:=TargetDir;
  Show;
  for  i:=0  to SourceFiles.items.Count-1 do
  begin
    try
      if FileExists(SourceFiles.items[i]) then
      begin
        Application.ProcessMessages;
        if zastavene then exit;
        AssignFile(Subor, SourceFiles.items[i]);
        Reset(Subor,1);
        Inc(VelkostSpolu,FileSize(Subor));
        CloseFile(subor);
      end;//if FileExists(SourceFiles.items[i]) then
    except
      raise;
    end;//except
 end;//
//      AssignFile(Subor, 'nic.txt');
 for  i:=0  to SourceFiles.items.Count-1 do
 begin
    if FileExists(SourceFiles.items[i]) then
    begin
      NewFileName:=TargetDir + '\' +
ExtractFileName(SourceFiles.items[i]);
    try
      AssignFile(NovySubor,NewFileName);
      AssignFile(Subor, SourceFiles.items[i]);
      Reset(Subor,1);
      Rewrite(NovySubor,1);
      ZatialPrekopirovane.Add(NewFileName);
      try
        odkial.Caption:=SourceFiles.items[i];
        kam.Caption:=NewFileName;
          while not Eof(subor) do
          begin
            Application.ProcessMessages;
            if zastavene then exit;
            BlockRead(subor,buffer,SizeOf(Buffer),BytesRead);
            NacitaneCelkom:=NacitaneCelkom + BytesRead;
            ProgressBar.Position:= round(NacitaneCelkom /
VelkostSpolu * 100);
            Application.ProcessMessages;
            if zastavene then exit;
            BlockWrite(NovySubor,buffer,BytesRead,BytesWritten);
          end;// while not Eof(subor) do
       finally
          CloseAllFiles;
          Handle:=FileOpen(SourceFiles.items[i],fmOpenRead);
          PovodnyDatum:=FileGetDate(handle);
          FileClose(Handle);
          Handle:=FileOpen(NewFileName,fmOpenReadWrite);
          FileSetDate(Handle,PovodnyDatum);
          FileClose(Handle);
        end;//finally
    except
      raise;
    end;//except
    end;//if FileExists(SourceFiles.items[i]) then
  end;//i:=0  to SourceFiles.items.Count-1 do
  finally
  ZatialPrekopirovane.Free;
  Close;
 end;//finally
end;
procedure TCopyForm.ZrusClick(Sender: TObject);
var i:integer;
begin
  zastavene:=true;
  CloseAllFiles;
  AllFilesClosed:=true;
  for i :=0  to ZatialPrekopirovane.Count-1 do
DeleteFile(ZatialPrekopirovane[i]);
end;
end.
```

že vybraná disková jednotka je naozaj prístupná. Na to potrebujeme napísať krátky kód, ktorý vložíme do obslužnej rutiny udalosti OnChange komponentu TDriveComboBox:

```
procedure TForm1.diskChange(Sender: TObject);
var tmp:string;
begin
  try
    tmp:=disk.Drive + ':\';
    ChDir(tmp);
    Dir.Drive:=disk.drive;
  except
   Application.MessageBox(Pchar('Jednotka ' + AnsiUpperCase(disk.drive)
+ ' je nedostupná!'),
    'Chyba', MB_OK OR MB_ICONERROR);
  end;// except
end;
```

Pomocníkom pri zisťovaní prístupnosti diskovej jednotky nám bude procedúra ChDir. Ako parameter jej odovzdáme názov jednotky a znaky „:\“. Inými slovami, pokúsime sa zmeniť aktuálny adresár na koreňový adresár danej diskovej jednotky. Ak uspejeme, vlastnosť Drive komponentu TDirectoryListBox nastavíme na jednotku, ktorú používateľ vybral pomocou komponentu TDriveComboBox. Ak pri pokuse o zmenu aktuálneho adresára vznikne výnimka, program jednoducho vypíše chybové hlásenie. Možno vás bude zaujímať nezvyčajné použitie slova pchar, ktoré sa používa pri deklaráciách premenných. Vysvetlenie je jednoduché: procedúre MessageBox potrebujeme odovzdať parameter typu pchar. Nám sa však nechce konvertovať, a tak sme reťazec typu string pretypovali na reťazec typu pchar:

```
Pchar('Jednotka ' + AnsiUpperCase(disk.drive) + ' je nedostupná!')
```

Pri pretypovaní najprv uvádzame cieľový dátový typ (v našom prípade pchar) a ako parameter uvádzame údaje, ktoré chceme pretypovať. Treba poznamenať, že používanie pretypovania si vyžaduje pomerne dobré znalosti OOP a jazyka Pascal. Okrem toho treba vedieť, že nie je možné pretypovať „všetko na všetko“, platia tu určité pravidlá. Nebudem ich však rozoberať, pretože presahujú rámec tohto článku. Pre úplnosť dodávam, že pretypovanie sa v angličtine označuje slovom typecast.

Kopírovanie súborov

Akiste vás zaujíma, ako v Delphi vytvoriť rutinu na kopírovanie súborov. Jednu celkom dobrú som pre vás napísal, keďže však ide o pomerne zložitý problém, rozhodol som sa poskytnúť vám kompletný zdrojový kód celej jednotky (unitu). Táto jednotka je de facto jednotkou formulára, ktorý bude zobrazovať priebeh kopírovania. Jednotka neobsahuje kód na spustenie kopírovania, ten si musíte vytvoriť sami vo vašej aplikácii. Samotné kopírovanie sa spúšťa takto:

```
CopyForm.TargetDir:='c:\copytarget';
CopyForm.StartCopy;
```

Obr. 2

TargetDir je názov cieľového adresára, do ktorého sa majú súbory prekopírovať. Samotný formulár na kopírovanie sa skladá z jedného komponentu TProgressBar, ďalej jedného komponentu TListBox a komponentov TLabel, ktoré nám ukazujú, ktorý súbor sa práve kopíruje. Komponent TListBox má vlastnosť Visible nastavenú na false, keďže nebude slúžiť ako vizuálny prvok, ale iba ako akýsi sklad názvov súborov, ktoré sa majú prekopírovať. Môžete ich špecifikovať priamo v Delphi (editovaním vlastnosti Items) alebo až za behu programu. Na prvý pohľad sa vám možno bude zdať zdrojový kód až priveľmi zložitý, ale nič sa nebojte, všetko si postupne preberieme a vysvetlíme. Pozorne si program odpíšte a môžeme začať. Prvá vec, ktorú si zrejme všimnete, bude použitie metódy Show. Pravdepodobne si kladiete otázku, prečo som nepoužil metódu ShowModal, veď ostatné prvky aplikácie by nemali byť počas kopírovania prístupné. Pravda je taká, že sa to dá urobiť aj touto metódou, kód však bude dosť zložitý a neprehľadný. Metóda Show totiž formulár iba zobrazí a kód sa vykonáva ďalej. Ak by som však použil metódu ShowModal, program by formulár iba zobrazil a potom by sa zastavil.

Ako som už povedal, samotné kopírovanie sa spustí vykonaním procedúry StartCopy. Na jej začiatku vidíme inicializácie premenných a „vynulovanie“ komponentu TProgressBar. Komponent TLabel s názvom Odkial potom zobrazí text „Zisťujem veľkosť súborov...“. Vaša prvá otázka zrejme bude, načo nám sú veľkosti súborov. Odpoveď je triviálna: pokiaľ chceme zobrazovať priebeh kopírovania, musíme vedieť, aká časť celku sa už skopírovala. A celok to je vlastne veľkosť všetkých súborov spolu. Výpočet celkovej veľkosti súborov sa vykonáva v cykle pomocou funkcie FileSize. Keďže objekt TStringList (od ktorého je odvodený komponent TListBox) začína číslovať prvky od nuly, aj cyklus začne „bežať“ od nuly. Pravdepodobne vás zaujíma, čo znamená záhadné Application.ProcessMessages. Tento príkaz zatiaľ necháme bez povšimnutia a vrátime sa k nemu neskôr. Ako poistka celého cyklu nám slúži funkcia FileExists, ktorá zisťuje, či daný súbor existuje, a až potom sa začne vykonávať zvyšok cyklu. Pokiaľ sa teda náhodou pomýlite a zadáte zlý názov súboru, tento súbor bude jednoducho vynechaný.

Nasledujúci obrovský cyklus má na starosti kopírovanie súborov. Pracuje na podobnom princípe, ako ten predchádzajúci: najskôr si overí, či daný súbor naozaj existuje, a až potom pokračuje ďalej. V prvej fáze sa z názvu a cesty zdrojového súboru vyextrahuje názov súboru. Meno cieľového súboru získame tak, že si vezmeme názov cieľového adresára a pridáme k nemu vyextrahovaný názov súboru. Procedúry AssignFile, Reset a Rewrite by vám mali byť známe a riadok

```
ZatialPrekopirovane.Add(NewFileName);
```

si vysvetlíme neskôr. Nasleduje nastavenie komponentov TLabel Odkial a Kam, ktoré nám zaistí zobrazenie informácie o tom, ktorý súbor sa práve kopíruje. Cyklus, ktorý sa začína riadkom

```
if FileExists(SourceFiles.items[i]) then...
```

má na starosti to, aby sa prekopírovali všetky súbory (inými slovami, všetky prvky komponentu TListBox). Teraz prejdeme do podcyklu hlavného cyklu. Tento podcyklus vykonáva samotné kopírovanie; začína sa riadkom

```
while not Eof(subor) do
```

Prečítaním tohto riadka sme sa práve zoznámili so sympatickou funkciou Eof, ktorá vráti true v prípade, že sa nachádzame na konci súboru a false v prípade, že tomu tak nie je. Jadro cyklu tvoria príkazy BlockRead a BlockWrite. V súvislosti s nimi vás teraz upozorním na jednu veľmi dôležitú vec, pozorne si prezrite tento riadok:

```
BlockWrite(NovySubor,buffer,BytesRead,BytesWritten);
```

Ak ste sa domnievali, že posledný parameter procedúry BlockRead (ktorý vracia počet skutočne načítaných bajtov) je zbytočný, ste na veľkom omyle. V našom prípade je totiž životne dôležitý. Prečo? Hneď si to vysvetlíme. Predstavme si napríklad, že kopírujeme veľmi malý súbor – len 10 bajtový a kopírujeme ho v blokoch po 8 bajtov. Keď cyklus prebehne prvýkrát, máme načítaných 8 bajtov a hneď ich aj zapíšeme do cieľového súboru. To však nestačí, keďže sme ešte nedosiahli koniec súboru, preto cyklus prebehne aj druhýkrát, a my za opäť pokúsime načítať osem bajtov. Bude to však dosť veľký problém, pretože nám ostali už len dva. Je logické, že nemôžeme zapísať (ani prečítať) osem bajtov, keď nám v zdrojovom súbore ostali neprečítané iba dva. Preto pri zápise (teda v procedúre BlockWrite) použijeme údaj o tom, koľko bajtov sme skutočne načítali.

Po zápise sa všetky súbory zatvoria pomocou procedúry CloseAllFiles (ostatné riadky vysvetlím neskôr). Nasledujú rutiny, ktoré zabezpečia, že prekopírovaný súbor bude mať ten istý dátum poslednej modifikácie ako súbor cieľový. Veď ako by to vyzeralo, keby všetky súbory, ktoré si dnes prekopírujeme, mali dnešný dátum. Najskôr sa zisťuje dátum poslednej modifikácie pôvodného súboru. K tomu je potrebné tento súbor otvoriť, čím získame jeho handle, ktorý potrebuje procedúra FileGetDate. Takto získaný dátum potom nastavíme aj cieľovému súboru a to prostredníctvom procedúry FileSetDate. Súbory je, samozrejme, potrebné zavrieť procedúrou CloseFile. Pokročilejší mi teraz dúfam odpustia malú odbočku, pokúsim sa začiatočníkom vysvetliť, čo je to handle.

Keď Windows otvára súbor, priradí mu určité jedinečné číslo (=handle), pomocou ktorého sa potom programátor na súbor odkazuje. Pri práci so súborom potom namiesto mena používate jeho jedinečný identifikátor – handle. Je potrebné poznamenať, že rutiny používajúce handle patria do kategórie nízkoúrovňových (low level) a v praxi iba málo používaných. Ako ale sami vidíte, ich použitie nie je žiadna veda, keď viete, ako nato.

Aby naša kopírovacia rutina nevyzerala príliš amatérsky, pridal som do nej možnosť prerušiť kopírovanie. Ak používateľ stlačí tlačítko Zrušiť, operácia sa preruší a všetky doteraz prekopírované súbory sa vymažú. Aby sme vedeli, ktoré súbory sa stihli prekopírovať, musíme si ich mená niekde evidovať. Zrejme už tušíte, že odkladať ich budeme do objektu ZatialPrekopirovane. Pokiaľ používateľ zruší kopírovanie, aktivuje sa cyklus, ktorý postupne prejde všetky prvky tohto objektu, ktorými sú vlastne názvy súborov, a všetky súbory postupne vymaže. Predtým ešte, samozrejme, uzavrieme všetky otvorené súbory pomocou procedúry CloseAllFiles. Aby však používateľ mohol stlačiť tlačítko Zrušiť, museli sme do kódu pridať riadok Application.ProcessMessages. Tento riadok zabezpečí, že riadenie na chvíľku preberie Windows (v skutočnosti je to trochu inak, túto formuláciu som použil kvôli jednoduchosti), ktorý prekreslí povrch dialógového okna a umožní stlačiť tlačítko. Potom sa aktivuje rutina, ktorá vymaže všetky doteraz prekopírované súbory. Je tu však ešte jeden problém: po vymazaní súborov bude program pokračovať presne tam, kde prestal, teda riadkom, ktorý nasleduje za Application.ProcessMessages. Aby sa to nestalo, museli sme zaviesť premennú Zastavene. Pokiaľ bude jej hodnota true, hlavný kopírovací cyklus sa preruší:

```
if zastavene then exit;
```

Po vykonaní príkazu Exit sa začne vykonávať blok `finally...end` a tým pádom sa aktivuje procedúra CloseAllFiles. A máme ďalší problém: prezrime si ešte raz metódu TCopyForm.ZrusClick. Ako vidíme, súbory sme už raz zatvorili a opätovný pokus o ich zatvorenie by spôsobil výnimku. Preto som zaviedol premennú AllFilesClosed, ktorá sa nastaví na true hneď ako používateľ preruší kopírovanie a súbory sa zatvoria. Procedúra CloseAllFiles najskôr zistí, či je táto premenná nastavená na true, a teda či sú súbory už zatvorené. Konečný výsledok nášho spoločného úsilia vidíme na obrázku č. 2.

Všimnime si usporiadanie blokov `try...except` a `try...finally`. Ako vidíme, je ich tu viac než dosť. Najdôležitejším z nich je blok začínajúci prvým príkazom try:

```
begin
try
.........
finally
ZatialPrekopirovane.Free;
Close;
end;//finally
```

Nech sa výnimka vyskytne kdekoľvek, tento blok ju vždy zachytí a uvoľní pamäť, ktorú zaberá objekt ZatialPrekopirovane. Je potrebné poznamenať, že množstvo `try...except` a `try...finally` blokov je značne závislé od toho, akú funkciu bude kopírovacia rutina plniť (napr. či bude slúžiť na kopírovanie súborov v nejakom súborovom manažéri alebo či bude slúžiť na zálohovanie). Okrem toho je možné počet týchto blokov zredukovať pomocou funkcie FileExists: pred otvorením súboru najskôr

zistíme, či ten naozaj existuje a ak áno, až potom sa vykonajú ostatné operácie. Pomerne veľké množstvo `try...except` a `try...finally` blokov v našom programe má skôr ilustračný ako praktický zmysel. Zamyslime sa nad týmto blokom:

```
for  i:=0  to SourceFiles.items.Count-1 do
 begin
    if FileExists(SourceFiles.items[i]) then
    begin
      NewFileName:=TargetDir + '\' +
ExtractFileName(SourceFiles.items[i]);
    try
      AssignFile(NovySubor,NewFileName);
      AssignFile(Subor, SourceFiles.items[i]);
      Reset(Subor,1);
      Rewrite(NovySubor,1);
.................
except
     raise;
end;//except
```

Existenciu súboru nám ešte pred jeho otvorením potvrdila funkcia FileExists, takže pri otváraní súboru s veľkou pravdepodobnosťou výnimka nevznikne. Ak by však predsa len vznikla (napríklad by sme ju spôsobili naschvál pokusom o otvorenie neexistujúceho súboru), začal by sa vykonávať blok `except...end`. To je zároveň aj dôvod, prečo táto sekvencia príkazov nie je umiestnená spolu s ostatnými príkazmi v jednom `try...finally` bloku. Ak by sme ju tam totiž umiestnili a pokúsili sa otvoriť neexistujúci súbor, vznikla by výnimka a začal by sa vykonávať kód `finally...end`, ktorý by sa pokúsil zatvoriť neexistujúci a neotvorený súbor.

Je dosť pravdepodobné, že problematike výnimiek neporozumiete ihneď. Pokiaľ ste s pochopením textu mali problémy, nič si z toho nerobte. Hlavne sa nebojte experimentovať, zo svojich vlastných pokusov niekedy získate viac, ako siahodlhým čítaním manuálov a helpov. Na záver „výnimkovania" by som si dovolil tvrdiť, že táto časť Delfínária bola výnimočná a to nielen svojou dĺžkou, ale aj množstvom informácií o výnimkách. A keďže výnimočný text môže často zapríčiniť vznik výnimiek, domyslite si na jeho začiatok slovko `try` a na jeho koniec slová `except raise end;` ☺

Nabudúce

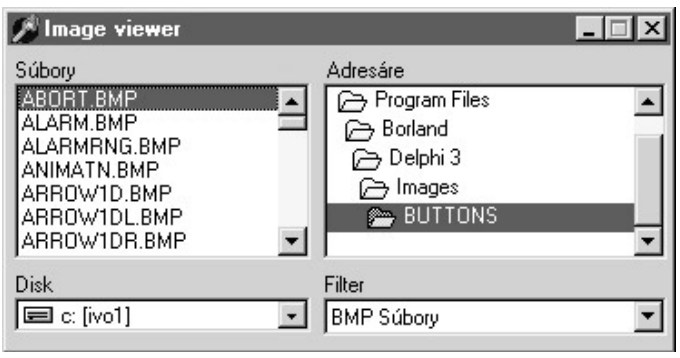
Témou budúcej časti bude grafika v Delphi. Zaoberať za budeme komponentom TImage, povieme si niečo o double bufferingu a ukážeme si praktické použitie niekoľkých Windows API funkcií. Priaznivcov 3D grafiky musím vopred upozorniť, že sa nebudeme venovať OpenGL ani DirectX . Pokiaľ vás táto problematika zaujíma, navštívte ľubovoľnú vyhľadávaciu službu a hľadajte slová „GLScene" (OpenGL rutiny) alebo „Delphi Games Creator" (DirectX komponenty pre Delphi). Na dnes je to všetko, dovidenia nabudúce.

Jedenásta časť: Grafika v Delphi

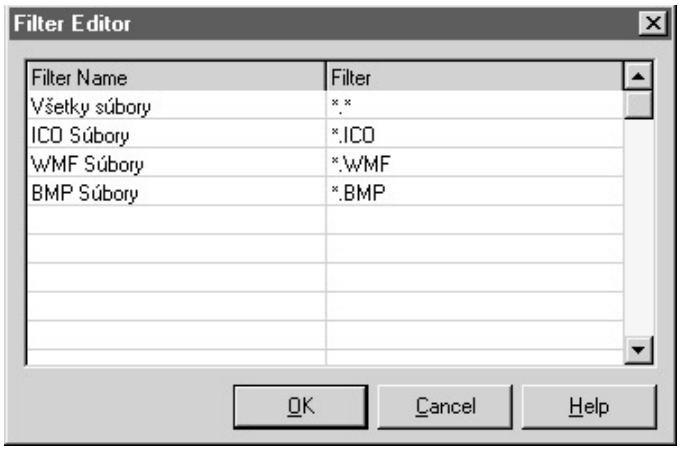
V tejto časti seriálu uvedieme niekoľko príkladov demonštrujúcich použitie grafiky v Delphi. Konkrétne pôjde o prehliadač obrázkov a jednoduchý kresliaci program. Potom si povieme niečo o double bufferingu a jeho význame a na záver si ukážeme použitie niektorých zaujímavých Windows API funkcií.

Jednoduchý prehliadač obrázkov

Prvým ukážkovým programom bude jednoduchý prehliadač obrázkov, schopný zobrazovať súbory BMP, ICO a WMF (pokiaľ hľadáte lepší komponent, navštívte Delphi super page a stiahnite si komponent TAdvImage). Základom bude používateľské rozhranie podobné programu EXE Info z predchádzajúcej časti seriálu (obr. 1). Formulár upravíme a pridáme komponent TFilterComboBox, ktorý nazveme Filter. Tento komponent nám bude slúžiť na filtrovanie súborov, ktoré má komponent TFileListBox zobraziť. Klikneme na vlastnosť Filter a vyplníme ju (obr. 2). Aby



Obr. 1



Obr. 2

komponent TFileListBox zobrazoval aktuálny typ súborov vybraných v komponente TFilterComboBox, je potrebná malá úprava:

```
procedure TForm1.FilterChange(Sender: TObject);
begin
  Files.Mask:=Filter.Mask;
end;
```

Ako vidíme, oba komponenty majú spoločnú vlastnosť, ktorou je Mask. Ak používateľ bude chcieť filtrovať súbory určitého typu, vyberie požadovaný druh súboru zo zoznamu Filter. Len čo to urobí, vyvolá sa obslužná rutina udalosti OnChange (komponentu TFilterComboBox) a práve zvolená maska sa priradí komponentu TFileListBox.

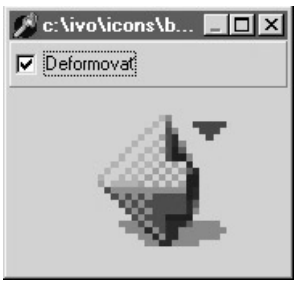
Keďže veľkosti obrázkov sú premenlivé, rozhodol som sa použiť na ich zobrazenie oddelený formulár. Do nášho projektu teda pridáme nový formulár a nazveme ho ImageForm. Rozmery tohto formulára sa budú nastavovať automaticky v závislosti od rozmerov obrázka. Je zrejmé, že tento postup nemožno použiť pri malých obrázkoch, pretože potom by sa nám do titulku okna nezmestil ani názov súboru. Preto je program napísaný tak, že najskôr nastaví rozmery formulára na 200 x 200 bodov. Ak je ktorýkoľvek rozmer obrázka väčší ako tieto implicitne nastavené rozmery, formulár sa automaticky zväčší. Samotný obrázok sa zobrazí dvojitým kliknutím na názov súboru:

```
procedure TForm1.FilesDblClick(Sender: TObject);
begin
try
  ImageForm.Image.Picture.LoadFromFile(files.FileName);
  ImageForm.Caption:=Files.FileName;
  ImageForm.Width:=200;
  ImageForm.Height:=200;
  if ImageForm.Image.Picture.width>200 then
    ImageForm.ClientWidth:=ImageForm.Image.Picture.Width;
  if ImageForm.Image.Picture.Height>200 then
    ImageForm.ClientHeight:=ImageForm.Image.Picture.Height;

  if ImageForm.Image.Picture.width>screen.Width then
     ImageForm.Width:=screen.Width;
  if ImageForm.Image.Picture.Height>screen.Height then
     ImageForm.Height:=Screen.Height;

  ImageForm.Position:=poScreenCenter;
  ImageForm.ShowModal;
except
  raise;
end;//except
end;
```

Ako vidíme, pri zmene veľkosti formulára sme nepoužili vlastnosti Width a Height, ale ClientWidth a ClientHeight. Dôvod je jednoduchý: vlastnosti Width a Height určujú veľkosť formulára vrátane titulkového pruhu a okrajovej čiary, ktorá lemuje formulár. My však potrebujeme zmeniť rozmer tzv. klientskej časti okna, t. j. tej časti okna, ktorú je možné použiť na umiestňovanie vizuálnych komponentov a kreslenie. ClientWidth a ClientHeight však nepoužijeme vtedy, keď je daný obrázok väčší ako naša obrazovka, vtedy sa totiž musí na obrazovku zmestiť hlavné okno spolu s titulkovým pruhom a okrajmi.

Pomocou vlastnosti Position umiestnime formulár do stredu obrazovky. Keby sme túto vlastnosť nastavili iba v Object Inspectore, formulár by sa síce umiestnil do stredu obrazovky a aj by tam ostal, ale iba dovtedy, kým by sme ním nepohli. Formuláre si totiž zapamätajú svoju polohu a vždy sa ukážu na tom mieste, na ktorom sa nachádzali, keď sme ich zatvorili.

Aby náš prehliadač nebol až taký jednoduchý, pridáme ešte možnosť deformácie obrázka. Na formulár ImageForm umiestnime komponent TPanel, vlastnosť Align nastavíme na alTop a pridáme zaškrtávacie pole, ktoré nazveme Deformovat. Pokiaľ bude toto pole zaškrtnuté, obrázok sa automaticky prispôsobí veľkosti formulára. Obslužná rutina udalosti OnClick zaškrtávacieho poľa bude vyzerať takto:



Obr. 3

```
procedure
TImageForm.DeformovatClick(Sender:
TObject);
begin
  Image.Stretch:=Deformovat.Checked;
end;
```

Ukážku deformovanej ikony vidíme na obrázku č. 3.

Jednoduchý grafický program

Druhým demo programom bude grafická aplikácia. Budete si v nej môcť vybrať z dvoch druhov štetca – štvorcového a kruhového –, pričom hrúbku štetca bude možné nastaviť. Ak sa vám tieto druhy štetcov budú zdať priveľké, budete môcť kresliť aj bod po bode. Okrem toho program umožní namiešať farbu, ktorá bude použitá pri tvorbe obrázka. Miešať ju budeme priamo zadávaním atribútov RGB. Aby sme s novou farbou nemuseli robiť skúšobné kresby, po zmene atribútov RGB sa novo namiešaná farba zobrazí v malom náhľadovom obdĺžniku.

Základným prvkom nášho programu bude opäť komponent TImage. Tentoraz však na formulári nebude sám, asistovať mu budú dva panely a niekoľko ďalších komponentov. Prvým krokom teda bude vytvorenie panelov. Prvý z nich bude obsahovať tlačidlá, ktoré nám umožnia vybrať si z dvoch typov štetca: kruhového a štvorcového. Nepôjde o tlačidlá typu TButton alebo TBitBtn, ako by ste sa možno nazdávali. Použijeme komponenty TSpeedButton, ktoré majú jednu pre nás veľmi potrebnú vlastnosť: schopnosť ostať stlačenými. Okrem toho sú vhodné aj z toho hľadiska, že automaticky ošetrujú „vyskočenie“ jedného tlačidla pri stlačení druhého. Pridáme teda tri tlačidlá TSpeedButton, prvé nazveme Stvorec, druhé Kruh a tretie Pixel. Aby tlačidlá jedno o druhom „vedeli“, musíme nastaviť vlastnosť GroupIndex na hodnotu 1.

Ďalšími dôležitými prvkami sú komponenty TSpinEdit, do ktorých budeme zapisovať hodnoty RGB. Nazveme ich rSpin, gSpin a bSpin. Potom vedľa nich pridáme jeden komponent TShape, ktorý nám bude slúžiť ako preview (ukážka) práve namiešanej farby (nazveme ho ColorTest). Napokon pridáme ešte ďalší komponent TSpinEdit, ktorý nazveme Hrubka. Ako už napovedá jeho názov, bude nám slúžiť na nastavovanie hrúbky čiary. Všetky tri komponenty TSpinEdit, ktoré majú na starosti nastavovanie hodnôt RGB, budú zdieľať jednu spoločnú rutinu, ktorú som nazval ColorsChanged:

```
procedure TPainterForm.ColorsChanged(sender:TObject);
var r,g,b:integer;
begin
  try
    r:=Integer(rSpin.Value);
    g:=Integer(gSpin.Value);
    b:=Integer(bSpin.Value);
    ColorTest.brush.color:=RGB(r,g,b);
    Image.canvas.brush.color:=RGB(r,g,b);
  except
  end;
end;
```

Túto procedúru v Object Inspectore nastavíme ako obslužnú rutinu udalosti OnChange všetkým trom komponentom TSpinEdit. Ako vidíme, hodnoty obsiahnuté v komponentoch TSpinEdit sú najskôr pretypované na typ integer a potom uložené do dočasnej premennej. Farbu výplne komponentu TShape určuje podvlastnosť vlastnosti Brush, ktorá má názov Color. Keďže táto vlastnosť je typu integer, potrebujeme funkciu, ktorá nám hodnoty RGB prekonvertuje na integer. Iste ste uhádli, že touto funkciou je funkcia RGB. Hneď ako dôjde ku zmene niektorého z parametrov RGB, zmení sa aj farba nášho malého preview obdĺžnika. Pri týchto operáciách pomerne často dochádza ku konvertačnej výnimke, nebudeme sa ňou však zaoberať, keďže neovplyvní funkčnosť nášho programu. Aby vás prípadné výnimky nezdržovali, vypnite voľbu Break on Exception (Tools – Environment Options).

Znalci angličtiny určite vedia, že slovo canvas znamená plátno. Pri kreslení na plátno musíme vedieť, akou farbou chceme kresliť. Je veľmi dôležité, aby ste program odpisovali pozorne a používali konštrukcie Image.Canvas, a nie iba Canvas. V opačnom prípade sa vám program síce skompiluje, no budete pracovať s plátnom hlavného formulára, a nie komponentu TImage. Ako je zrejmé z výpisu programu, farbu, ktorou budeme kresliť, nastavíme pomocou vlastnosti Color. Najdôležitejšou v celom programe je obsluha udalosti OnMouseMove komponentu TImage:

```
procedure TPainterForm.ImageMouseMove(Sender: TObject; Shift:
TShiftState;
  X, Y: Integer);
begin
  image.Canvas.Pen.style:=psClear;
  if mdown then
  begin
    if elipsa.Down then
      image.Canvas.Ellipse(
      x+hrubka.Value,y + hrubka.Value,x,y);
    if Obdlznik.Down then
      image.Canvas.rectangle(
      x+hrubka.Value,y + hrubka.Value,x,y);
    if Pixel.Down then
      Image.Canvas.Pixels[x,y]:=
      image.canvas.brush.color;
  end;//if mdown then
end;
```

Keďže kreslenie sa realizuje iba vtedy, keď je stlačené ľavé tlačidlo myši, museli sme zaviesť pomocnú premennú s názvom mdown. Deklarovaná je hneď za kľúčovým slovom implementation. Táto premenná sa nastaví na true, pokiaľ bolo stlačené ľavé tlačidlo myši:

```
procedure TPainterForm.ImageMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  if Button=mbLeft then MDown:=true;
end;
```

Informáciu o tom, ktoré tlačidlo myši bolo stlačené, zisťujeme pomocou parametra Button. Je zrejmé, že po pustení tlačidla treba nastaviť túto premennú na false:

```
procedure TPainterForm.ImageMouseUp(Sender: TObject; Button:
TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  MDown:=false;
end;
```

Vráťme sa však k procedúre, ktorá má na starosti samotné kreslenie. To sa realizuje kreslením malých kruhov (pomocou metódy Ellipse) alebo obdĺžnikov (pomocou metódy Rectangle). Veľkosť týchto útvarov je ovplyvnená údajom z komponentu Hrubka. Aby okolo týchto útvarov nebola vykresľovaná obrysová čiara, nastavili sme vlastnosť Style objektu Pen na psClear. Samozrejme, kresliť je možné aj po jednotlivých bodoch – pixloch. V tom prípade použijeme vlastnosť Pixels. Je však potrebné dodať, že pokiaľ budeme kresliť pomocou tejto vlastnosti, parameter hrúbka nebude mať na kreslenie nijaký vplyv. Pixel je totiž základnou zobrazovacou jednotkou, ktorej veľkosť je nemenná.

Aby náš program nebol až taký jednoduchý, rozhodol som sa pridať do neho možnosť otvárania a ukladania BMP súborov. Nebudem tu podrobne opisovať tvorbu menu, pokiaľ si neviete poradiť, pozrite si druhú časť seriálu. Zameriam sa iba na výpisy procedúr. Na otváranie obrázkov slúži komponent TOpenPictureDialog. Ten sa od komponentu TOpenDialog líši tým, že pred otvorením obrázka zobrazí malý náhľad. Jeho použitie je veľmi jednoduché:

```
procedure TPainterForm.OtvoritClick(Sender: TObject);
begin
  if OpenPic.Execute then
  image.Picture.LoadFromFile(OpenPic.filename);
  ColorsChanged(sender);
end;
```

Po otvorení súboru sa musí vykonať procedúra ColorsChanged, aby sa mohla vlastnosť Brush nastaviť na základe našich hodnôt RGB. Aby sme mohli naše expresionistické veľdiela aj ukladať, pridal som možnosť ukladania súborov:

```
procedure TPainterForm.UlozClick(Sender: TObject);
begin
  if SavePic.Execute then
  Image.Picture.SaveToFile(SavePic.FileName);
end;
```

Každý slušný program poskytuje možnosti práce so schránkou. Pridáme preto menu Schránka, ktoré bude obsahovať dve položky: Kopírovať a Prilepiť. Grafiku obsiahnutú v komponente TImage vložíme do schránky veľmi jednoducho:

```
procedure TPainterForm.kopirovatClick(Sender: TObject);
begin
  Clipboard.Assign(Image.picture);
end;
```

Kompletný výpis metódy StartClick:

```
procedure TForm1.StartClick(Sender: TObject);
var I:integer;
StredX,StredY:integer;
Strana:integer;
Five:integer;
Buffer:TBitmap;
label zaciatok;
begin
try
  Buffer:=TBitmap.Create;
  Buffer.Width:=two.Width;
  Buffer.Height:=Two.Height;
  zaciatok:
  with one.canvas do
  begin
    Brush.Color:=ClWhite;
    Rectangle(0,0,Width,Height);
  end;//with
  with Buffer.canvas do
  begin
    Brush.Color:=ClWhite;
    Rectangle(0,0,Width,Height);
  end;//with
    Five:=0;
    for i :=20  to 90 do
    begin
      if ukoncene then exit;
      if five=5 then five:=0;
      inc(five);
      Strana:=i;
      if five=5 then
      begin
        StredX:=(one.Width div 2) - Strana div 2;
        StredY:=(one.Height div 2) - Strana div 2;
        one.Canvas.Brush.Color:=clBlack;
        Buffer.Canvas.Brush.Color:=one.Canvas.Brush.Color;
         one.Canvas.FrameRect(Rect(StredX,StredY,StredX+Strana,StredY
+ Strana));
       Buffer.Canvas.FrameRect(Rect(StredX,StredY,StredX+Strana,StredY
+ Strana));
two.Canvas.CopyRect(Rect(0,0,Two.Width,Two.Height),Buffer.Canvas,Rect(0,0,Buffer.Width,Buffer.Height));
        Application.ProcessMessages;
      end;//if five=5 then
    end;//for
  goto zaciatok;
finally
  buffer.Free;
end;//finally
end;
```

Trošku zložitejšie bude prilepenie objektu zo schránky:

```
procedure TPainterForm.PrilepitClick(Sender: TObject);
var
  Bitmap: TBitmap;
begin
 if Clipboard.HasFormat(CF_BITMAP) then
 begin
    Bitmap := TBitmap.Create;
   try
     Bitmap.Assign(Clipboard);
     Image.Canvas.Draw(0, 0, Bitmap);
   finally
     Bitmap.Free;
   end;//finally
end;//if Clipboard.HasFormat(CF_BITMAP) then
end;
```

Táto procedúra pomocou metódy HasFormat objektu Clipboard najskôr zistí, či je v schránke uložený objekt typu bitmapa. Ak áno, vytvorí sa v pamäti dočasná bitmapa (objekt TBitmap), do ktorej sa pomocou metódy Assign prekopíruje obsah schránky. Z dočasnej bitmapy sa potom obrázok nakreslí na plátno (objekt Canvas) komponentu TImage pomocou metódy Draw. Poslednou funkciou, ktorú do nášho programu pridáme, bude vytvorenie nového obrázka:

```
procedure TPainterForm.NovyClick(Sender: TObject);
begin
  image.picture.graphic.Width:=PainterForm.ClientWidth;
  image.picture.graphic.Height:=PainterForm.ClientHeight;
  with Image.Canvas do
  begin
    brush.color:=clWhite;
    brush.Style:=bsSolid;
    pen.Style:=psClear;
    Rectangle(
    0,0,PainterForm.ClientWidth,
    PainterForm.ClientHeight);
  end;//with
  ColorsChanged(sender);
end;
```

Obr. 4

Pre jednoduchosť som program napísal tak, že sa nás na rozmery nového obrázka neopýta, ale preberie ich z veľkosti hlavného formulára. Na tomto mieste je dôležité poznamenať, že pri zmene veľkosti obrázka je potrebné nastavovať vlastnosti Width a Height objektu Graphic, a nie komponentu TImage. Vymazanie obrázka sa realizuje pomocou metódy Rectangle: na celú šírku a výšku klientskej časti formulára jednoducho nakreslíme biely obdĺžnik. Nakoniec prebehne inicializácia parametrov RGB – vykoná sa procedúra ColorsChanged. Konečnú podobu programu vidíme na obrázku č. 4.

Double buffering

Znalcom počítačových hier bude tento termín akiste známy. Pre tých, ktorí hry pričasto nehrali, tu mám vysvetlenie. Pri počítačových hrách zohrávala nesmiernu úlohu rýchlosť, preto boli väčšinou napísané v assembleri. Šikovní programátori hier vymysleli veľa trikov, aby ich produkty bežali čo najrýchlejšie a najplynulejšie. Jedným z nich je double buffering. V podstate ide o to, že nekreslíme priamo na obrazovku, ale do vyrovnávacej pamäte – bufera. Len čo sme s kreslením hotoví, obsah vyrovnávacej pamäte presunieme na obrazovku. Táto technika pomáha odstrániť jav známy ako „trhanie“ (po anglicky flicker). Hneď na úvod musím povedať, že tento trik nemusí byť pod Windows ani zďaleka taký efektívny, ako bol pod DOS-om. Hlavný problém spočíva v tom, že rýchlosť niektorých častí Windows GDI je z počítača na počítač premenlivá. Je možné, že práve na vašom stroji bude efektívnejšie double buffering nepoužiť. Rozhodne však nebude na škodu, keď o ňom budete vedieť.

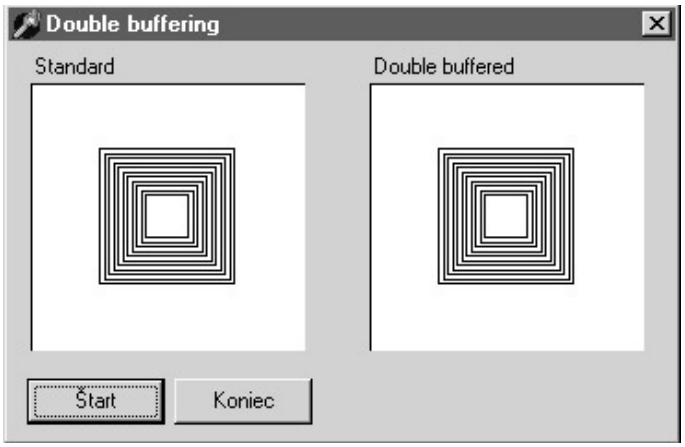
Keďže problematika double bufferingu sa v nemalej miere dotýka počítačových hier, dovolil by som si na tomto mieste krátku úvahu na tému Delphi a hry. V minulosti sa hry písali prevažne v assembleri, prípadne v jazyku C. Prakticky všetky operácie boli z dôvodu rýchlosti realizované na úrovni priameho prístupu k hardvéru. Žartovne by sme mohli povedať, že hry s počítačom robili, čo chceli. Pod Windows je však situácia iná: „šéfom“ je Windows, ktorý hry doslova obmedzuje. Preto Microsoft vyvinul rozhranie DirectX, ktoré pomocou funkcií API umožňuje pristupovať priamo k hardvéru. Toto rozhranie je naozaj efektívne iba vo Windows 95, pretože Windows NT priamy prístup k hardvéru neumožňuje. Ak by ste v Delphi chceli písať hry, museli by ste použiť funkcie z DirectX alebo iného rozhrania (napr. OpenGL alebo Glide). To isté platí pre programy s podporou 3D akcelerátorov: musíte mať k dispozícii dokumentáciu výrobcu príslušnej karty a opis API. Nestačí sa jednoducho spoliehať, že váš Delphi program bude kresliť rýchlejšie, pretože máte v stroji 3D kartu. Pokiaľ chcete tvoriť naozaj kvalitné 3D aplikácie, použite OpenGL. Funguje pod Windows NT a podľa mojich informácií by malo fungovať aj pod Windows 95 OSR2 a Windows 98. Pokiaľ vás OpenGL zaujíma, navštívte stránku www.opengl.org. Ak ste zvedaví na konkrétnu implementáciu v Delphi, navštívte stránku Delphi super page a hľadajte komponent GLScene od Mikea Lischkeho. Pokiaľ vás zaujíma DirectX, hľadajte na internete slová „Delphi games creator“.

Vráťme sa však späť k double bufferingu. Náš ukážkový program sa bude skladať z dvoch komponentov TImage (prvý nazveme One a druhý Two), umiestnených vedľa

seba. Jeden z nich bude vykresľovať jednoduchú animáciu štandardným spôsobom a druhý pomocou double bufferingu. Okrem toho budú na formulári dve tlačidlá: jedno na štart animácie a druhé na ukončenie programu. Metóda StartClick, ktorá slúži na realizáciu animácie, je na prvý pohľad pomerne zložitá. Jej kompletný výpis nájdete v rámčeku.

Princíp práce je jednoduchý: najskôr vytvoríme objekt typu bitmapa. Potom mu priradíme rovnaké rozmery, ako má komponent Two. Nasleduje inicializácia: pomocou metódy Rectangle nakreslíme biely obdĺžnik na plochu komponentu One a aj na našu neviditeľnú bitmapu. Potom nasleduje cyklus, ktorí vykreslí každý piaty štvorec. Ak by sme totiž vykresľovali všetky štvorce, dostali by sme vyplnený útvar. Výpočty nie sú príliš podstatné, preto ich nebudem bližšie komentovať. Po nich nasleduje nastavenie vlastnosti Brush, ktorá ovplyvní činnosť metódy FrameRect. Možno si teraz kladiete otázku, prečo sme namiesto metódy FrameRect nepoužili radšej metódu Rectangle. Pravda je taká, že je možné použiť aj túto metódu, ale výsledný efekt bude trochu iný. Rozdiel medzi metódami FrameRect a Rectangle je v tom, že FrameRect kreslí iba obrysovú čiaru a nevyplňuje útvar, ktorý táto čiara obopína. Metóda Rectangle, naopak, útvar vyplní, čo sme napokon aj využili pri vymazávaní obsahu komponentu TImage. Pozornejší z vás si akiste všimli použitie funkcie Rect pri odovzdávaní parametrov metóde FrameRect. Táto metóda totiž preberá parametre typu TRect. Aby ste nemuseli zbytočne vytvárať premennú toho typu, vytvoríme ju „za jazdy" prostredníctvom funkcie Rect.

Nasleduje najdôležitejšia časť programu: prekopírovanie obsahu bufera do komponentu Two. Na tento účel nám slúži metóda CopyRect. Treba podotknúť, že ju musí volať



Obr. 5

ten objekt, na ktorého plátno (canvas) sa má daná obdĺžniková oblasť prekopírovať. Prvým parametrom sú súradnice cieľového miesta, kam bude prekopírovaný obsah zo zdrojovej oblasti. Ako vidíme, aj tu nám príde vhod použitie funkcie Rect, ušetríme si zbytočné deklarácie. Druhý parameter je typu TCanvas a udáva zdrojové plátno, z ktorého budeme kopírovať. Tretí parameter udáva oblasť zdrojového plátna, ktorá bude prekopírovaná do cieľovej oblasti. Poistkou je blok try...finally, ktorý nám uvoľní vyrovnávaciu pamäť v prípade, že počas behu programu nastane výnimka. Pokiaľ ste pozorne čítali predchádzajúcu časť seriálu, určite viete, na čo nám slúži pomocná premenná ukoncene a aj metóda ProcessMessages. Obe slúžia ako pomocné nástroje na to, aby náš program mohol byť ukončený. Metóda TForm1.StartClick sa ukončí, pokiaľ bude táto pomocná premenná nastavená na true. Pred ukončením programu ju teda treba nastaviť na true:

```
procedure TForm1.QuitClick(Sender: TObject);
begin
  ukoncene:=true;
  Application.Terminate;
end;
```

Výsledok vidíme na obrázku č. 5.

Udalosť OnPaint

V nasledujúcich odsekoch si povieme niečo o črtách grafického subsystému Windows. Tí z vás, ktorí sa dokážu ako-tak orientovať v systéme Windows správ (Windows messages), môžu tieto odseky pokojne preskočiť. Hneď na úvod musím upozorniť, že nebudem vysvetľovať princíp práce s Windows správami. Pokročilejší mi, dúfam, odpustia množstvo zjednodušení, ktorých sa dopustím, aby som problém priblížil aj začiatočníkom.

Dajme sa teda do práce a zamyslime sa nad nasledujúcim problémom: na obrazovke máme okno. Aby to nebolo také jednoduché, toto okno bude do polovice prekryté iným oknom. Hneď ako toto druhé okno zmizne, treba prekresliť plochu prvého okna. Nie veľmi efektívnym riešením by bolo, keby si systém uložil do pamäte plochu, na ktorú sa vykreslilo druhé okno, a po jeho zmiznutí by takto zrekonštruoval pôvodný vzhľad prvého okna. Je však nad slnko jasnejšie, že takéto riešenie by spotrebovalo nesmierne veľa pamäte. Našťastie vývojári Windows prišli s niečím oveľa efektívnejším. Ako som už naznačil, nechcem sa ponárať do detailov okolo Windows správ, preto vám budem toto druhé riešenie ilustrovať formou vtipného dialógu. Ten sa začne v momente, keď používateľ chytí, prípadne zatvorí druhé okno a prvé okno je potrebné prekresliť. Prosím pozor, nasleduje dialóg:

- **Windows**: „Počuj, okienko, používateľ práve uzavrel okno, ktoré bolo nad tebou. Prosím ťa, láskavo sa prekresli.
- **"Okienko**: „Yes, sir!"

Takýto postup je jednoduchý a zároveň veľmi účinný. Okno, keďže sa vie samo nakresliť, vie sa aj samo prekresliť a netreba si jeho obsah zapamätávať. Windows plní úlohu akéhosi organizátora: sleduje, kedy ktoré okno sa pohlo a kedy treba ktoré okno prekresliť.

Tieto tvrdenia vám dokážem na malom príklade. Ukážkový program nakreslí vyplnený štvoruholník, elipsu a ľubovoľný N-uholník. Vytvoríme preto formulár, ktorý bude obsahovať komponent TEdit, do ktorého napíšeme počet uhlov, ktoré má N-uholník mať. Samozrejme, nesmieme zabudnúť pridať aj tlačidlo, ktoré nám procedúru spustí. Kreslenie kruhu a štvoruholníka má na starosti procedúra TForm1.Kresli:

```
procedure TForm1.Kresli;
begin
  Canvas.Brush.Color:=clBlack;
  Canvas.Brush.Style:=bsDiagCross;
  Canvas.Rectangle(10,10,80,80);

  Canvas.Brush.Color:=clBlack;
  Canvas.Brush.Style:=bsFDiagonal;
  Canvas.Ellipse(90,10,190,80);
end;
```

Túto metódu budeme volať v rámci druhej metódy, ktorá nám nakreslí N-uholník:

```
procedure TForm1.NakresliClick(Sender: TObject);
const PI=3.14159;
var PocStran,i:integer;
PolyArray: array[0..15] of TPoint;
begin
  Canvas.Brush.color:=Color;
  Canvas.Pen.Style:=psClear;
  Canvas.Rectangle(0,0,Width,100);
  Canvas.Pen.Style:=psSolid;
  Kresli;

  Canvas.Brush.Style:=bsClear;
  PocStran:=StrToIntDef(Strany.Text,3);
  for i :=0  to PocStran do
  begin
     PolyArray[i]:=
      Point(Trunc(Sin((2*PI)*i/PocStran)*30) + 240,
            Trunc(Cos((2*PI)*i/PocStran)*30) + 40);
  end;//for i :=0  to PocStran do
  for  i:=PocStran + 1  to 15 do PolyArray[i]:=PolyArray[0];
  Canvas.Polyline(PolyArray);
end;
```

Táto metóda najskôr pomocou metódy Rectangle vymaže určitú časť formulára:

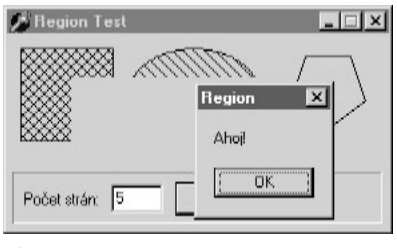
```
Canvas.Rectangle(0,0,Width,100);
```

Číslo 100 pravdepodobne budete musieť vo vašom programe nahradiť iným číslom, aby sa nevymazal celý formulár, ale iba tá časť, na ktorú sa vykresľujú naše útvary. Teraz malé odbočenie: všimnite si riadok

```
Canvas.Brush.Color:=Color;
```

Tento riadok znamená to isté, ako keby sme napísali

```
Canvas.Brush.Color:=form1.Color;
```



Obr. 6

Pokiaľ explicitne neuvedieme (napr. príkazom with alebo priamym identifikovaním komponentu či objektu), na ktorý komponent sa odvolávame, Delphi automaticky predpokladá, že sa odvolávame na formulár. Samotný výpočet nie je dôležitý, rád by som však ešte upozornil na funkciu IntToStrDef. Tá sa pokúsi skonvertovať daný reťazec na typ integer. Pokiaľ neuspeje, nevyvolá výnimku, ale použije implicitnú (default) hodnotu, v našom prípade číslo 3. Preto ak do komponentu TEdit nenapíšeme nič, program nám nakreslí trojuholník. Vráťme sa však späť k pôvodnej téme. Po napísaní programu napíšme nasledujúcu rutinu:

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

```
  ShowMessage('Ahoj!');
end;
```

Kliknite na formulár a malé dialógové okienko s nápisom „Ahoj!“ po ňom trochu poposúvajte. Budete svedkami nepríjemného efektu, podobného tomu na obrázku č. 6. Po odsunutí okna zistíme, že naše útvary buď zmizli, alebo ostali „ohlodané“. Je to spôsobené prekresľovaním: všetky komponenty VCL sa prekre-sľujú automaticky. Komponent TForm vie iba toľko, že došlo k „narušeniu“ určitej oblasti formulára, tak ju prekreslí farbou, ktorá je nastavená vo vlastnosti Color. O našich útvaroch nevie nič, preto ich neprekreslí. Musíme sa teda o ne postarať sami. Našťastie riešenie je veľmi jednoduché: vždy, keď formulár dostane správu o prekreslení, vyvolá sa udalosť OnPaint. Nám teda stačí volanie našej metódy umiestniť do obslužnej rutiny tejto udalosti:

```
procedure TForm1.FormPaint(Sender: TObject);
begin
  NakresliClick(Sender);
end;
```

Malý trik s Windows API

Porozmýšľajte teraz chvíľu, ako by ste riešili nasledujúci problém: máte obdĺžnik, a ak naň používateľ klikne myšou, treba mu to oznámiť. Riešenie je pomerne jednoduché: najskôr musíte zistiť, kam používateľ klikol. Na základe týchto údajov potom pokračujete ďalej: pokiaľ tieto body patria do oblasti ohraničenej štvorcom, používateľ naň určite klikol. Čo však v prípade, ak chceme náš program prerobiť tak, aby používateľovi oznámil, či klikol na štvoruholník, elipsu alebo mnohouholník? Na pomoc nám prichádzajú funkcie Windows API. Skôr než ich použijeme, musíme najprv deklarovať potrebné premenné, a to hneď za kľúčovým slovom implementation, aby ich bolo „vidieť“ v celej jednotke:

```
implementation
var
PolyRgn,EllipseRegion,RectRegion:HRgn;
```

Program si najskôr upravíme, vyskúšame a nakoniec si vysvetlíme, ako pracuje. Zmien naozaj nebude veľa, preto nebudem ešte raz uvádzať celý výpis procedúry, ale iba riadky, ktoré treba pripísať, plus riadok, za ktorý treba pripísať nový príkaz. Prvé zmeny vykonáme v procedúre TForm1.Kresli:

```
Canvas.Rectangle(10,10,80,80);
RectRegion:=CreateRectRgn(10,10,80,80);
.......................
Canvas.Ellipse(90,10,190,80);
EllipseRegion:=CreateEllipticRgn(90,10,190,80);
```

Ďalšia zmena sa bude týkať metódy Form1.NakresliClick:

```
Canvas.Polyline(PolyArray);
PolyRgn:=CreatePolygonRgn(PolyArray,15,WINDING)
```

Potom je potrebné pridať samotný kód, ktorý zistí, či došlo ku kliknutiu na niektorý z našich útvarov:

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
    if PtInRegion(RectRegion,x,y) then
        Application.MessageBox('Klikli ste na štvoruholník.','Oznam',
MB_OK OR MB_ICONINFORMATION);

    if PtInRegion(EllipseRegion,x,y) then
           Application.MessageBox('Klikli ste na elipsu.','Oznam',
MB_OK OR MB_ICONINFORMATION);

    if PtInRegion(PolyRgn,x,y) then
        Application.MessageBox('Klikli ste na N - uholník.','Oznam',
MB_OK OR MB_ICONINFORMATION);
end;
```

Napokon je potrebné objekty typu HRgn uvoľniť z pamäte:

```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
 DeleteObject(RectRegion);
 DeleteObject(EllipseRegion);
 DeleteObject(PolyRgn);
end;
```

Jadrom programu je Windows API funkcia PtInRegion, ktorá zisťuje, či sa bod so súradnicami X a Y nachádza v danej oblasti. Oblasť je reprezentovaná dátovým typom HRgn. Aby systém Windows „vedel“, ktorú oblasť máme na mysli, musíme túto oblasť najskôr vytvoriť. V prípade, že ide o štvoruholník, použijeme funkciu CreateRectRgn. Parametre pritom majú rovnaký význam ako pri metóde Rectangle. Podobne to je s funkciou CreateEllipticRgn. Trošku odlišný postup sa musí použiť v prípade mnohouholníka. Keďže ide o útvar s niekoľkými vrcholmi, funkcia CreatePolygonRgn potrebuje mať zoznam všetkých vrcholov (prvý parameter) a aj ich počet (druhý parameter). Tretí parameter vysvetľovať nebudem, pretože presahuje rámec tohto článku (a okrem toho by si vysvetlenie vyžadovalo nemalé znalosti Windows API). Je zrejmé, že premenné typu HRgn zaberajú určité množstvo pamäte (v skutočnosti je to trochu inak, túto formuláciu som použil pre jednoduchosť), preto ich treba pred ukončením programu uvoľniť:

Obr. 7

```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
 DeleteObject(RectRegion);
 DeleteObject(EllipseRegion);
 DeleteObject(PolyRgn);
end;
```

Finálnu verziu programu vidíme na obrázku č. 7.

Nabudúce

Ak sa vám Delfinárium páčilo, mám pre vás jednu smutnú správu: nasledujúca časť bude záverečná. Zameriame sa v nej na multimédiá: povieme si niečo o systémových zvukoch Windows a napíšeme si jednoduchý audioprehrávač a CD prehrávač.

Literatúra:
D. Osier – S. Grobman – S. Batson: Teach yourself Delphi 2 in 21 days.

Záver seriálu: Multimédiá

V záverečnej časti seriálu si ukážeme, ako v praxi vyzerá podpora multimédií v Delphi. Hneď na úvod treba povedať, že sa budeme zaoberať iba štandardnými komponentmi dodávanými s Delphi. Tento vývojový nástroj má však veľké množstvo priaznivcov na celom svete a tí vytvorili mnoho šikovných komponentov, ktoré vám uľahčia vývoj vašej multimediálnej aplikácie. Ak teda máte prístup k internetu, navštívte Delphi super page a vyberte stránku Multimedia, určite neoľutujete.

Úvod do multimédií

Grafika a multimédiá nepochybne patria medzi najatraktívnejšie témy z oblasti programovania. Pohľad na databázovú aplikáciu poteší jedine odborníka na databázy, medicínske programy upútajú iba lekárov, ale multimédiá oslovia každého, kto sa o počítače aspoň trošku zaujíma. Ich príťažlivosť spočíva predovšetkým v tom, že dokážu pritiahnuť dva najdôležitejšie zmysly človeka: zrak a sluch. Kvalitná audiovizuálna prezentácia môže niekedy dokonca navodiť také pocity, aké prežívate pri sledovaní dobrého filmu. Tí z vás, ktorí poznajú demoskupinu Future Crew, určite vedia, o čom hovorím. A keďže sú multimédiá veľmi príťažlivé, každý kvalitný vývojový nástroj ich nejakým spôsobom podporuje. Na tomto mieste treba podotknúť, že možnosti poskytované vývojovými nástrojmi sa vám zrejme po krátkom čase budú zdať obmedzené. Je to z toho dôvodu, že tieto nástroje používajú vysokoúrovňové rozhranie MCI (Media Control Interface). Pokiaľ chcete napísať kvalitný prehrávač skladieb, neostane vám nič iné ako siahnuť po Windows API funkciách. Ak by vám ani tie nestačili, môžete využiť špeciálne rozhranie DirectX (konkrétne jeho časť DirectSound), ktoré bolo vyvinuté práve na podporu multimédií a hier. Skôr ako sa pustíte do písania zložitých rutín, prehľadajte internet, či náhodou niekto váš problém už nevyriešil. A ešte jedna drobná rada: ak píšete multimediálny program, nezabudnite do USES klauzuly pridať jednotku mmsystem. Táto jednotka obsahuje veľké množstvo funkcií a konštánt, ktoré sa vám pri písaní programov zídu.

Zvuky Windows

Windows priniesli okrem nového používateľského rozhrania aj niekoľko príjemných prekvapení pre majiteľov zvukových kariet. Umožnili totiž k rôznym systémovým udalostiam priradiť rôzne zvuky. Počítač tak konečne prestal mlčať a stal sa prítulnejším. My si teraz napíšeme krátky program, ktorý demonštruje použitie týchto zvukov. Najskôr je však potrebné vedieť zoznam systémových zvukov, ktoré môžeme prehrávať:

MB_ICONASTERISK	Zvuk hviezdička
MB_ICONEXLAMATION	Zvuk protest

MB_ICONHAND	Zvuk kritického prerušenia
MB_ICONQUESTION	Zvuk otázka
MB_OK	Zvuk implicitného pípnutia

Náš program bude pozostávať zo zoznamu týchto zvukov a jednoduchého príkazu, ktorý nám ich prehrá. Vytvoríme teda formulár, pridáme komponent TComboBox, komponent TBevel a dve tlačidlá (obr. 1). Kód na prehrávanie implicitného zvuku je veľmi jednoduchý:

```
procedure TForm1.BeepClick(Sender: TObject);
var BeepItem:Cardinal;
begin
  case sounds.ItemIndex of
    0: BeepItem:=MB_ICONASTERISK;
    1: BeepItem:=MB_ICONEXCLAMATION;
    2: BeepItem:=MB_ICONHAND;
    3: BeepItem:=MB_ICONQUESTION;
    4: BeepItem:=MB_OK;
  end;
  MessageBeep(BeepItem);
end;
```

Treba povedať, že v praxi takýto postup nemusíte používať, pretože prehranie zvuku sa realizuje automaticky v závislosti od parametrov, ktoré odovzdáte procedúre MessageBox. Nebude však na škodu, ak budete vedieť, že existuje aj takáto možnosť.

Obr. 1

Komponent TMediaPlayer

V Delphi budete drvivú väčšinu multimediálnych operácií realizovať prostredníctvom komponentu TMediaPlayer. Tento komponent vám poskytuje jednoduché používateľské rozhranie s niekoľkými tlačidlami, reprezentujúcimi základné multimediálne funkcie. Vzhľad programu vidíme na obrázku č. 2, skladá sa z audioprehrávača a CD prehrávača. Najskôr pridáme na formulár dva panely, prvý z nich bude nositeľom ostatných panelov a v druhom sa bude zobrazovať video. Čierne pozadie nápisov je vlastne takisto komponent TPanel, ktorý má nastavenú čiernu farbu (t. j. vlastnosť Color:=clBlack). Vlastnosti BevelInner a BevelOuter sú nastavené na bvNone. Text je vlastne komponent typu TLabel, ktorého farba pozadia (vlastnosť Color) je nastavená na čiernu a farba písma je nastavená na žltú. Komponent TLabel s nápisom „Prehráva sa“ pomenujeme PrehravaSa. Tento komponent nám bude zobrazovať názov súboru (bez cesty), ktorý sa práve prehráva. Komponent roztiahneme a vlastnosť AutoSize nastavíme na flase. Tým zabezpečíme, že názov súboru nám „nepretečie“ za okraj panela. Komponent TLabel s nápisom „Dĺžka“ nazveme Dlzka, bude nám zobrazovať dĺžku súboru (nie v bajtoch, ale v jednotkách závislých od typu súboru).

Je nad slnko jasnejšie, že spôsob zobrazovania dĺžky je pri každom type média iný. Pri prehrávaní WAV súborov sa zobrazuje dĺžka audiosekvencie v milisekundách, pri práci s AVI súbormi je zase dĺžka definovaná ako celkové množstvo políčok (frames), ktoré AVI súbor obsahuje. Pre jednoduchosť som tieto údaje konvertoval jedine v prípade WAV súborov, kde je dĺžka vlastne dĺžkou trvania zvuku v milisekundách a konverzia na sekundy veľmi jednoduchá. Pokiaľ s takýmto riešením nie ste spokojní, vyhľadajte si v helpe vlastnosť TimeFormat komponentu TMediaPlayer. Pomocou nej môžete získať údaje o dĺžke média v požadovanom formáte. Musím vás však upozorniť, že si to bude vyžadovať veľa experimentovania. Z toho istého dôvodu som sa rozhodol použiť rovnaký spôsob zobrazovania pri zobrazovaní informácie o tom, aká časť súboru sa už prehrala. Tú nám bude indikovať komponent TLabel, ktorý nazveme Pozicia. Obsah tohto komponentu sa bude obnovovať dosť často (každých 250 ms), preto musí byť text v ňom čo najkratší. Vyriešil som to tak, že do vlastnosti Caption tohto komponentu som napísal nulu a naľavo som umiestnil druhý komponent TLabel, v ktorom je nápis „Pozícia:“. Takto sa blikanie zredukuje na minimum.

Súbor sa otvára malým tlačidlom typu TSpeedButton, ktoré som nazval OpenBtn. Ďalším komponentom na formulári je komponent TTrackBar, ktorý nám bude zobrazovať priebeh prehrávania príslušného súboru. Vlastnosť Orientation tohto komponentu nastavíme na trHorizontal, LineSize nastavíme na 2, Frequency na 10, TickMarks na tmBottomRight a Max nastavíme na 100. Potom pridáme políčko s nápisom „Auto play“ a nazveme ho AutoPlay. Ďalšie políčko bude obsahovať text „Video v okne“ a bude sa volať Video. Na otváranie súborov bude, samozrejme, potrebný komponent TOpenDialog. Filter súborov vyplníme podľa obrázka č. 3.

Aby sme vedeli zistiť, aká časť súboru sa už prehrala, musíme tento údaj získavať v určitých časových intervaloch. Preto na formulár pridáme komponent TTimer, ktorý sa bude aktivovať v intervale 250 ms. Pomenujeme ho StatusTimer a vlastnosť Active nastavíme na false. Obslužná rutina udalosti OnTimer nám pomôže zistiť, aká časť súboru sa už prehrala:

```
procedure TMegaPlayer.StatusTimerTimer(Sender: TObject);
begin
 if MediaOne.Mode=mpPlaying then
   begin
    Pozicia.Caption:=
    IntToStr(MediaOne.Position);
    TrackBar.Position:=
    Round(MediaOne.Position / MediaOne.Length *100);
  end;//if MediaOne.Mode=mpOpen then
end;
```

Pomocou vlastnosti Mode najskôr zistíme, či je komponent v režime prehrávania, a ak áno, pokračujeme ďalej. Prvú časť výstavby formulára ukončíme pridaním komponentu TMediaPlayer, ktorý nazveme MediaOne. Ako vidíme, obsahuje mnoho tlačidiel, z ktorých však budeme potrebovať iba niekoľko. Zoznam viditeľných tlačidiel nám umožňuje nastaviť vlastnosť VisibleButtons. Na hodnotu true nastavíme tieto tlačidlá: btPlay, btPause, btStop, btStep a btBack. Vlastnosť DeviceType nastavíme na dtAutoSelect, komponent totiž na základe prípony súboru sám určí metódu prehrávania. Súbor, ktorý si chceme prehrať, vyberáme stlačením tlačidla OpenBtn. Obslužná rutina udalosti OnClick tohto tlačidla má nasledujúcu podobu:

```
procedure TMegaPlayer.OpenBtnClick(Sender: TObject);
begin
  if Open.Execute then
  begin
    InitMediaFile(Open.FileName);
    if autoplay.Checked then
    begin
       MediaOne.Perform(WM_LBUTTONDOWN,0,$00090009);
       MediaOne.Perform(WM_LBUTTONUP,0,$00090009);
    end;//if autoplay.Checked then
  end;//if
end;
```

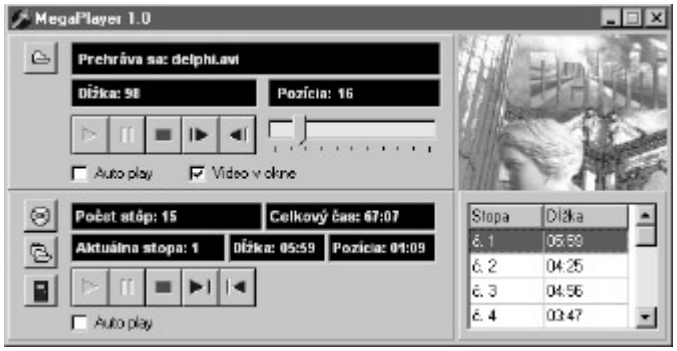
Ako vidíme, najskôr sa zobrazí dialógové okno, ktoré nám umožní zvoliť si súbor. Potom pomocou procedúry InitMediaFile vykonáme inicializáciu. Pokiaľ je zaškrtnuté políčko AutoPlay, súbor sa začne automaticky prehrávať. Na spustenie prehrávania som použil metódu Perform, ktorá simuluje stlačenie tlačidla Play. Prehrávanie je možné spustiť aj metódou Play, potom však nebudú korektne povolené a zakázané tlačidlá komponentu TMediaPlayer. Teraz si ukážeme, ako prebieha inicializácia:

```
procedure TMegaPlayer.InitMediaFile(what: string);
begin
  try
    Screen.Cursor:=crHourGlass;
    with MediaOne do
    begin
     FileName:=what;
     PrehravaSa.Caption:='Prehráva sa: '+ ExtractFileName(what);
     if what='' then exit;
     Close;
     Open;
     Dlzka.Caption:='Dĺžka: '+IntToStr(Length);
   end;//with MediaOne do
  finally
    Screen.Cursor:=crDefault;
  end;
end;
```

Priradením parametra what vlastnosti FileName nastavíme názov súboru, ktorý bude komponent TMediaPlayer prehrávať. Potom sa z tohto súboru odstráni cesta a jeho názov bude zobrazovať komponent TLabel s názvom PrehravaSa. Metódou Close najskôr uzavrieme súbor, ktorý bol predtým otvorený, a potom zavoláme metódu Open. Informáciu o časovej dĺžke súboru získame pomocou vlastnosti Length komponentu TMediaPlayer.

Už vieme, že komponent TTimer sme použili preto, aby sme mohli v určitých časových intervaloch získavať informácie o priebehu prehrávania. Jeho vlastnosť Enabled sme však nastavili na false, aby zbytočne nevolal obslužnú rutinu, keď program práve neprehráva nijaký súbor. Preto je tento komponent potrebné aktivovať ručne, a to konkrétne po stlačení tlačidla Play. Po stlačení tlačidla Stop ho zase treba deaktivovať:

```
procedure TMegaPlayer.MediaOneClick(Sender: TObject; Button:
TMPBtnType;
  var DoDefault: Boolean);
begin
    if button=btPlay then
    StatusTimer.Enabled:=true;
    if button=btStop then
    StatusTimer.Enabled:=false;
end;
```



Obr. 2

Ak ste už pracovali s komponentom TMediaPlayer, určite už viete, že implicitne si pri prehrávaní videa vytvorí vlastné okno, ale je možné jeho výstup presmerovať na ľubovoľného potomka triedy TWinControl. Preto som na pravý okraj formulára umiestnil panel, na ktorom budem túto schopnosť komponentu demonštrovať. Kľúčový kód sa nachádza v obslužnej rutine udalosti OnClick komponentu Video:

```
procedure TMegaPlayer.videoClick(Sender: TObject);
begin
  if video.Checked then
  begin
    MediaOne.Display:=DisplayPanel;
  end
  else
  begin
    MediaOne.Display:=nil;
    DisplayPanel.Invalidate;
  end;//else
  InitMediaFile(mediaOne.FileName);
end;
```

Cieľový komponent, v ktorom sa bude video zobrazovať, nastavíme pomocou vlastnosti Display. Pokiaľ si to rozmyslíme a chceme prehrávať AVI súbory opäť vo zvláštnom okne, túto vlastnosť nastavíme na nil. Aby nám na paneli neostali „zvyšky" videa, prekreslíme ho pomocou metódy Invalidate. Celá vec má však jeden háčik: zmena vlastnosti Display sa odrazí až po zatvorení a otvorení súboru. Tento problém som vyriešil tak, že program vykoná inicializáciu, pričom sa použije názov toho istého súboru, ktorý sa práve prehrával. Takto nemusíme súbor otvárať ešte raz ručne. Keď program spustíme prvýkrát, vlastnosť FileName ešte nebude obsahovať názov súboru, bude prázdna. Preto aj parameter odovzdaný procedúre InitMediaFile bude obsahovať prázdny reťazec. Pozrite si ešte raz pozorne rutinu InitMediaFile. Ako vidíte, počítal som so všetkým a umiestnil som do nej riadok:

```
if what='' then exit;
```

Vďaka tomuto riadku nenastane situácia, keď by sa komponent TMediaPlayer pokúsil otvoriť súbor, ktorého názov je vlastne prázdny reťazec.

Aby náš program nebol taký triviálny, rozhodol som sa rozšíriť ho o podporu drag and drop. Nebudeme teda musieť súbory otvárať ručne, súbor jednoducho pretiahneme z Explorera alebo Windows Commandera. Do USES klauzuly pridáme jednotku ShellAPI a pustíme sa do práce. Aby sme do nášho programu mohli integrovať podporu drag and drop, musíme použiť niektoré funkcie Windows API. Pokročilejší mi, dúfam, odpustia, že sa opäť pokúsim technické detaily vysvetliť čo najjednoduchšie, aby som začiatočníkom zbytočne neplietol hlavu. V prvom rade musí operačný systém vedieť handle okna, ktoré bude podporovať drag and drop (inými slovami, musí poznať handle okna, ktoré bude prijímať WM_DROPFILES správy). Aby sme si to zbytočne nekomplikovali, napíšeme program tak, aby sa dali súbory „hodiť" kamkoľvek na formulár. Preto použijeme handle formulára:

```
procedure TMegaPlayer.FormShow(Sender: TObject);
begin
  DragAcceptFiles(Handle, True);
end;
```

Samozrejme, toto samo osebe nestačí. Nasledovať bude procedúra, ktorá zistí, či sme na formulár pretiahli multimediálny súbor. Ak áno, komponent TMediaPlayer tento súbor otvorí:

```
procedure TMegaPlayer.WMDropFiles(var Msg: TWMDropFiles);
var
  CFileName: array[0..MAX_PATH] of Char;
  tmp:string;
begin
  try
    if DragQueryFile(
    Msg.Drop, 0, CFileName, MAX_PATH) > 0 then
    begin
    tmp:=CFileName;
    tmp:=AnsiLowerCase(tmp);
    if (pos('.wav',tmp)>0) or
   (pos('.avi',tmp)>0)
    or (pos('.mid',tmp)>0) then
    begin
      InitMediaFile(tmp);
      if autoplay.Checked then
      begin
        MediaOne.Perform(WM_LBUTTONDOWN,0,$00090009);
        MediaOne.Perform(WM_LBUTTONUP,0,$00090009);
      end;//if autoplay.Checked then
    end;//if pos('.wav',tmp)>0....
      Msg.Result := 0;
    end;
  finally
    DragFinish(Msg.Drop);
  end;
end;
```

Názov súboru získame pomocou funkcie DragQueryFile. Pri prenose názvu súboru sa spotrebuje určité množstvo pamäte. Túto pamäť uvoľníme pomocou funkcie DragFinish. Audioprehrávač je teda hotový, pustime sa teraz do výroby CD prehrávača.

CD prehrávač

Rozhranie MCI podporuje okrem niektorých zvukových formátov aj prácu s jednotkou CD-ROM. A keďže komponent TMediaPlayer používa služby MCI, dokáže s mechanikou CD-ROM takisto pracovať. Doteraz tento komponent rozoznával typy súborov sám, a to podľa ich koncovky. Tentoraz mu však budeme musieť trochu pomôcť, keďže audio CD nie je súbor, nemá koncovku, a teda musíme vlastnosť DeviceType nastaviť ručne na dtCDAudio. Ešte než sa pustíme do práce na CD prehrávači, musím sa s vami podeliť o jeden veľmi nepríjemný zážitok, ktorý sa mi stal pri jeho písaní.

Pri tvorbe programu som vychádzal z príkladov skvelej knihy Mistrovství v Delphi 2. Autor v nej použil mnoho užitočných rutín z knižnice mmsystem, pomocou ktorých zisťoval celkovú dĺžku CD disku, dĺžky jednotlivých skladieb atď. Komponent TMediaPlayer v tomto smere poskytuje slušné možnosti, problém je však s dekódovaním údajov, ktoré vám poskytne. V mojom prípade išlo doslova o dekódovanie, pretože rutiny opísané v uvedenej knihe jednoducho nefungovali. Sám autor v jednej vete naznačil, že komponent TMediaPlayer neposkytuje údaje o čase v takom formáte, aký sme mu nastavili. Nazrel som teda do nápovede, no ani to nepomohlo. Základný princíp získavania údajov bol síce vysvetlený správne, ibaže nefungoval. Neostávalo mi teda nič iné ako vyriešiť problém metódou pokus – omyl, čo sa mi nakoniec aj podarilo. Nemôžem však zaručiť, že uvedené rutiny budú správne pracovať aj vo verziách vyšších ako 3. Delphi 4, bohužiaľ, nemám k dispozícii, a tak som program nemohol vyskúšať. Ako nočná mora možno bude znieť tvrdenie, že v prípade, ak sa vám podarí tento problém vyriešiť pod Delphi 4, nie je isté, či bude fungovať pod Delphi 5... Rozhodol som sa vám preto poskytnúť niekoľko indícií, ktoré by vám mali pomôcť v ťažkostiach. Hlavný problém spočíva v tom, že komponent TMediaPlayer nevypĺňa vlastnosti pracujúce s časom (napr. Position) tak, ako by mal; nedodržiava formát, ktorý mu nastavíme. Príslušný formát je potrebné stanoviť pomocou vlastnosti TimeFormat. Pokiaľ je táto vlastnosť nastavená napríklad na hodnotu tfTMSF, mal by vlastnosti pracujúce s časom vracať údaje vo formáte Tracks, Minutes, Seconds, Frames. Počas písania ukážkového programu som zistil, že v skutočnosti sú dáta uložené v nasledujúcom formáte:

```
type
  MSRec = record
    Mins: byte;
    Seconds: byte;
    NotUsed1: byte;
    NotUsed2: byte;
  end;
```

Šokujúce je, že údaje o celkovej dĺžke CD síce možno získať, pri prehrávaní CD sú však údaje uložené zase iným spôsobom a jeden údaj je dokonca schovaný v poli NotUsed1. Pokiaľ teda budete mať problémy, neostane vám nič iné, ako zistiť si „polohu" príslušného údaja ručne. Použite pritom takýto postup: najskôr deklarujte premennú typu record, podobnú uvedenej. Potom z nej pomocou pretypovania extrahujte jednotlivé údaje:

```
with MSRec(CurTrackLength) do
  begin
    AktualnaStopa.Caption:=
    IntToStr(CurTrack);
    DlzkaStopy.Caption:=Format(
    '%.2d:%.2d',[Mins, Seconds]);
  end;//with MSRec(CurTrackLength) do
```

Ako vidíme, vďaka pretypovaniu sme dosiahli, že k premennej CurTrackLength, ktorá je typu longint, môžeme pristupovať tak, ako keby to bola premenná typu MSRec. Potom premennú musíte skúmať bajt po bajte a zistiť, ktorý z nich obsahuje hľadané údaje. Pevne verím, že tento návod nebudete potrebovať...

Vráťme sa však k nášmu prehrávaču. Ako vidíme, v podstate sa veľmi nelíši od audioprehrávača. Prv než začneme písať kód, preberieme si mená jednotlivých komponentov, z ktorých sa skladá. Program nám po vložení CD disku zobrazí celkový počet stôp. Tento údaj sa bude zobrazovať v komponente TLabel, ktorý nesie názov PocetStop. Ďalší komponent TLabel bude zobrazovať celkovú dĺžku CD v minútach a sekundách, preto som ho nazval CelkovyCas. Tento komponent však nezobrazuje nápis „Celkový čas“, ako by sa na prvý pohľad mohlo zdať. Keďže tento údaj sa bude často prekresľovať, umiestnil som do neho iba tri otázniky, ktoré budú viditeľné pri prvom spustení programu. Samotný nápis „Celkový čas“ je umiestnený v druhom komponente TLabel, ktorý je tesne vedľa komponentu CelkovyCas. Podobne je to aj s komponentmi AktualnaStopa, DlzkaStopy a CDPozicia. Všetky sú vyplnené otáznikmi, vedľa ktorých sa nachádzajú príslušné nápisy. Pri opakovanom prekresľovaní sa tak nebude prekresľovať celý text, ale iba jeho nevyhnutná časť. Hneď na úvod musím povedať, že nie všetky riešenia, ktoré som vo svojom programe použil, možno považovať za ideálne. Ako príklad by mohol poslúžiť práve problém s blikaním často prekresľovaných textov. V skutočnosti treba opakovane prekresľovať iba jediný údaj: pozíciu, na ktorej sa práve nachádzame. Celková dĺžka CD sa totiž nemení a dĺžky jednotlivých skladieb takisto nie. Pre jednoduchosť som však použil toto menej efektívne riešenie.

Tri bočné tlačidlá nesú nasledujúce názvy: EjectCD, OpenMedia a ZavriCD. Pomocou prvého tlačidla budeme môcť otvoriť CD-ROM mechaniku, pomocou druhého inicializujeme samotný prehrávač po vložení nového CD a pomocou tretieho tlačidla uzavrieme komponent TMediaPlayer. Zatvorenie komponentu je potrebné z toho dôvodu, aby mechaniku CD-ROM mohli používať aj iné programy. Pokiaľ by sme toto tlačidlo vynechali, museli by sme náš program najskôr zavrieť, až potom by bola CD-ROM mechanika dostupná pre ostatné aplikácie. Najdôležitejším prvkom je komponent TMediaPlayer, ktorý som nazval MediaTwo. Podobne ako audioprehrávač aj CD prehrávač nám umožní spustiť prehrávanie hneď po vložení CD a stlačení tlačidla OpenMedia. Komponent umiestnený úplne napravo patrí do čeľade komponentov TStringGrid a nesie názov TrackGrid. V ňom sa budú zobrazovať údaje o číslach skladieb a dĺžke ich trvania.

Základný opis programu teda máme za sebou, teraz príde tá ťažšia časť – programovanie. Ako prvý si ukážeme modifikovaný výpis obsluhy udalosti OnShow hlavného formulára, ktorý nám zinicializuje stĺpce komponentu TStringGrid a pridá do nich nadpisy:

```
procedure TMegaPlayer.FormShow(Sender: TObject);
begin
  DragAcceptFiles(Handle, True);
  TrackGrid.Cells[0,0]:='Stopa';
  TrackGrid.Cells[1,0]:='Dĺžka';
end;
```

Samotná práca s aplikáciou sa začína vložením CD disku a stlačením tlačidla OpenMedia. Ukážme si teda, čo sa pod týmto tlačidlom skrýva:

```
procedure TMegaPlayer.OpenMediaClick(Sender: TObject);
begin
  try
   try
    Screen.Cursor:=crHourGlass;
    MediaTwo.TimeFormat:=tfTMSF;
    MediaTwo.Open;
    Application.ProcessMessages;
    if MediaTwo.Tracks>100 then
    begin
     MediaTwo.Close;
     Application.MessageBox('Prosím vložte CD.',
     'MegaPlayer',MB_OK OR MB_ICONEXCLAMATION);
     exit;
    end;
    PocetStop.Caption:='Počet stôp: '+
    IntToStr(MediaTwo.Tracks);
    with MSRec(MediaTwo.Length) do
    begin
      CelkovyCas.Caption:=Format(
      '%.2d:%.2d',[mins,seconds]);
    end;//with
    FillTrackGrid;
    if CDautoplay.Checked then
    begin
       MediaTwo.Perform
       (WM_LBUTTONDOWN,0,$00090009);
       MediaTwo.Perform
       (WM_LBUTTONUP,0,$00090009);
    end;//if autoplay.Checked then
   except
    On E:Exception do
    begin
      if MediaTwo.Error=291 then
      Application.MessageBox
      ('CD ROM mechaniku používa iný program.',
      'Chyba',MB_OK OR MB_ICONERROR)
      else raise;
    end;
   end;//except
   finally
    Screen.Cursor:=crDefault;
  end;//finally
end;
```

Nebudem rutinu opisovať príliš detailne, zameriam sa iba na najdôležitejšie časti. Po vykonaní metódy Open treba zistiť, či mechanike vôbec nejaký disk je. Problém je v tom, že komponent TMediaPlayer aj bez disku veselo pokračuje v práci a program sa zacyklí. Našťastie tomuto nepríjemnému úkazu je možné zabrániť. Stačí, keď použijeme vlastnosť Tracks, ktorá nám vracia počet stôp. Pokiaľ v mechanike nie je vložené CD, táto vlastnosť nám vráti veľmi veľké číslo. Ja som sa rozhodol použiť ako hranicu číslo 100, pokiaľ teda táto vlastnosť vráti väčší počet stôp ako 100, program vás požiada o vloženie CD disku. Predtým však pre istotu zatvorí komponent TMediaPlayer, aby sme ani náhodou nestlačili tlačidlo Play.

Pokiaľ je počet stôp menší ako 100, komponent PocetStop sa nastaví podľa vlastnosti Tracks. Nasleduje kód, ktorý pomocou pretypovania získa z vlastnosti Length údaje o dĺžke média v minútach a sekundách. Samotné pretypovanie je vykonané pomocou príkazu

```
with MSRec(MediaTwo.Length) do....
```

Najskôr uvádzame požadovaný dátový typ a potom údaj, ktorý má byť pretypovaný. Všimnime si, že sme funkcii Format ako parametre odovzdali údaje Mins a Seconds. Pozrime sa teraz ešte raz na deklaráciu dátového typu MSRec. Ako vidíme, pristupovali sme vlastne k štruktúram tohto dátového typu. Inými slovami, s vlastnosťou Length komponentu TMediaPlayer sme pracovali tak, ako keby išlo o premennú typu MSRec. Funkcia Format nám zabezpečí, že údaj o dĺžke CD sa vypíše vo formáte 00:00, t. j. vždy sa použijú dve číslice (napr. po vložení CD s dĺžkou 70 minút a 3 sekundy bude program ukazovať 70:03).

Obr. 3

Nasledujúca procedúra FillTrackGrid prejde všetky stopy CD disku a získané údaje umiestni do komponentu TStringGrid. Výpis tejto procedúry si ukážeme čoskoro, najskôr si však na udržanie kontinuity vysvetlíme význam príkazov, ktoré za ňou nasledujú. Ako vidíme, ide o sekvenciu príkazov, ktoré zisťujú, či je zaškrtnuté políčko CDAutoPlay. V prípade, že toto políčko zaškrtnuté je, program pomocou metódy Perform nasimuluje kliknutie myšou na súradnice 9 a 9. Nezabúdajte, že nemôžeme použiť metódu Play, pretože potom nebudú tlačidlá komponentu TMediaPlayer správne povolené a zakázané. Nasleduje obsluha výnimky, ktorá sa aktivuje v prípade, že CD-ROM mechaniku už používa iný program. Vráťme sa však k procedúre FillTrackGrid. Jej výpis vyzerá takto:

```
procedure TMegaPlayer.FillTrackGrid;
var i:integer;
CurTrackLength,CurTrack:integer;
begin
 try
  TrackGrid.RowCount:=MediaTwo.Tracks+1;
  for  i:=1  to MediaTwo.Tracks do
  begin
   MediaTwo.Position:=i;
   TrackGrid.Cells[0,i]:='č. '+ IntToStr(i);
   CurTrack:=
   MCI_TMSF_TRACK(MediaTwo.Position);
   CurTrackLength:=
   MediaTwo.TrackLength[CurTrack];
   with MSRec(CurTrackLength) do
   begin
   TrackGrid.Cells[1,i]:=Format(
   '%.2d:%.2d',[Mins, Seconds]);
   end;//with MSRec(CurTrackLength) do
  end;//for}
 finally
  MediaTwo.Position:=1;
```

```
 end;//finally
end;
```

Opäť začíname vlastnosťou Tracks, pomocou ktorej nastavíme počet riadkov v komponente TStringGrid. Tento komponent však musí mať o jeden riadok viac, ako je počet stôp na disku, pretože jeden riadok zaberajú nadpisy stĺpcov. Nasleduje cyklus, ktorý prejde postupne všetky stopy disku a zistí ich veľkosť. Zo stopy na stopu sa premiestňujeme prostredníctvom vlastnosti Position. Dĺžku stopy získame pomocou vlastnosti TrackLength, ktorej ako parameter odovzdáme číslo stopy, ktorej dĺžku chceme zistiť. Nasleduje pretypovanie a formátovanie, pomocou ktorého premeníme nečitateľné čísla, získané pomocou vlastnosti TrackLength, na zmysluplné údaje. Nakoniec sa pomocou vlastnosti Position opäť premiestime na začiatok disku. Disk teda vložený máme, údaje o stopách takisto, diskotéka sa teda môže začať. Ani tentoraz sa však pri zisťovaní údajov o priebehu prehrávania nevyhneme použitiu komponentu TTimer. Pripomínam, že ho potrebujeme preto, aby sme v určitých časových intervaloch vedeli zistiť, koľko minút a sekúnd sa už z práve prehrávanej skladby odohralo. Obslužná rutina udalosti OnClick komponentu MediaTwo je podobná rutine, ktorú sme použili pri komponente MediaOne:

```
procedure TMegaPlayer.MediaTwoClick(Sender: TObject; Button:
TMPBtnType;
  var DoDefault: Boolean);
begin
   if button=btPlay then
   StatusTimer.Enabled:=true;
   if button=btStop then
   StatusTimer.Enabled:=false;
 end;
```

Aby náš kód správne fungoval, musíme modifikovať aj rutinu časovača:

```
procedure TMegaPlayer.StatusTimerTimer(Sender: TObject);
begin
   if MediaOne.Mode=mpPlaying then
   begin
    Pozicia.Caption:=IntToStr(MediaOne.Position);
    TrackBar.Position:=
    Round(MediaOne.Position / MediaOne.Length *100);
  end;//if MediaOne.Mode=mpOpen then
  if MediaTwo.Mode=mpPlaying then
  UpdateCDStatus;
end;
```

Údaje o tom, aká časť skladby je už za nami, nám zobrazuje procedúra UpdateCDStatus:

```
procedure TMegaPlayer.UpdateCDStatus;
var CurTrackLength,CurTrack:integer;
begin
  CurTrack:=
  MCI_TMSF_TRACK(MediaTwo.Position);
  CurTrackLength:=
  MediaTwo.TrackLength[CurTrack];

  with MSRec(CurTrackLength) do
  begin
    AktualnaStopa.Caption:=
    IntToStr(CurTrack);
    DlzkaStopy.Caption:=Format(
    '%.2d:%.2d',[Mins, Seconds]);
  end;//with MSRec(CurTrackLength) do
  with MSRec(MediaTwo.Position) do
  CDPozicia.Caption:=Format(
  '%.2d:%.2d',[Seconds, NotUsed1]);
```

Najskôr potrebujeme zistiť číslo prehrávanej stopy. Tentoraz však nemôžeme použiť priamo vlastnosť Position, pretože CD sa už točí a táto vlastnosť obsahuje okrem čísla stopy aj informácie o pozícii v rámci práve prehrávanej skladby. Číslo stopy preto musíme vyextrahovať pomocou funkcie MCI_TMSF_TRACK. Potom zistíme dĺžku aktuálnej stopy. Nasleduje formátovanie údajov a aktualizácia príslušných komponentov TLabel. Pomocou údajov získaných z vlastnosti Length sa zistí dĺžka prehrávanej piesne. Až potiaľto neboli žiadne problémy. Človek by predpokladal, že v prípade zisťovania pozície v rámci piesne sa použije ten istý postup. Rozdiel by mal byť jedine v tom, že namiesto pretypovania premennej CurTrackLength pretypujeme vlastnosť Position. Bohužiaľ, realita bola iná. Počet minút, ktoré ubehli od začiatku prehrávania piesne, nie je umiestnený v štruktúre Mins, ale Seconds. Sekundy zasa nájdeme tam, kde by sme ich hľadali najmenej: v štruktúre NotUsed1. Pripomínam, že všetky tieto poznatky som získal metódou pokus – omyl, pretože dokumentácia v tomto prípade zlyhala. Prehrávanie je možné zastaviť pomocou tlačidla Stop. Pokiaľ prehrávanie neprerušíme, bude pokračovať dokonca aj po ukončení programu. Aby sme tomu zabránili, musíme prehrávanie počas ukončovania aplikácie automaticky zastaviť:

```
procedure TMegaPlayer.FormClose(Sender: TObject; var Action:
TCloseAction);
begin
  if MediaOne.Mode=mpPlaying then MediaOne.Stop;
  if MediaTwo.Mode=mpPlaying then MediaTwo.Stop;
  MediaOne.Close;
  MediaTwo.Close;
end;
```

Dúfam, že ste nezabudli na tlačidlo EjectCD, pomocou ktorého vieme otvoriť dvierka CD-ROM mechaniky. Po jeho stlačení sa vykoná nasledujúci kód:

```
procedure TMegaPlayer.EjectCDClick(Sender: TObject);
begin
  if mpCanEject in MediaTwo.Capabilities then
  begin
    with MediaTwo do
    begin
      if MediaTwo.Mode=mpPlaying then
      stop;
      StatusTimer.Enabled:=false;
      eject;
      Close;
      TrackGrid.RowCount:=1;
    end;//with
  end;//if
end;
```

Na začiatku procedúry najskôr zisťujeme, či dané zariadenie (čiže CD-ROM) podporuje funkciu Eject. Pokiaľ ju CD-ROM podporuje, prerušíme prehrávanie, zastavíme komponent TTimer (pretože sme prerušili prehrávanie, nepotrebujeme získavať stavové informácie), uzavrieme komponent TMediaPlayer a napokon mechaniku otvoríme metódou Eject. Potom zrušíme zoznam skladieb: vlastnosť RowCount nastavíme na 1 a ostanú nám iba nadpisy oboch stĺpcov. Podotýkam, že metóda Eject funguje iba vtedy, ak je komponent TMediaPlayer otvorený. Podobný postup používa aj rutina napojená na tlačidlo ZavriCD: ak chceme dať mechaniku k dispozícii iným aplikáciám, musíme prehrávanie prerušiť a komponent TMediaPlayer zatvoriť:

```
procedure TMegaPlayer.ZavriCDClick(Sender: TObject);
begin
  if MediaTwo.Mode=mpPlaying then MediaTwo.Stop;
  MediaTwo.Close;
  TrackGrid.RowCount:=1;
end;
```

Keďže je komponent TMediaPlayer zatvorený, nemá zmysel zobrazovať zoznam skladieb, preto ho vymažeme. A propos zoznam skladieb: aby nebol neužitočný, rozhodol som sa našu aplikáciu modifikovať tak, aby sa po dvojitom kliknutí na zoznam začala prehrávať príslušná skladba:

```
procedure TMegaPlayer.TrackGridDblClick(Sender: TObject);
begin
  if MediaTwo.mode=mpPlaying then MediaTwo.Stop;
  MediaTwo.Position:=TrackGrid.Row;
  MediaTwo.Perform(WM_LBUTTONDOWN,0,$00090009);
  MediaTwo.Perform(WM_LBUTTONUP,0,$00090009);
  MediaTwo.Play;
end;
```

V prípade, že sa ešte prehráva iná skladba, musíme prehrávanie prerušiť pomocou metódy Stop. Číslo skladby je vlastne číslo riadka, v ktorom sa údaj o skladbe nachádza. Aby boli všetky tlačidlá správne zakázané a povolené, opäť som použil metódu Perform. Keďže CD-ROM mechanika má pomerne dlhé ohlasy, nie je isté, že v čase vykonávania metódy Perform bude tlačidlo Play dostupné. Inými slovami, môže sa stať, že nasimulujeme kliknutie na nepovolené (disabled) tlačidlo Play. Preto som pre istotu pridal aj metódu Play, s ktorou bude všetko dokonale fungovať. Tak, a sme hotoví. Naša cesta do sveta multimédií sa končí. Dúfam, že sa vám cestovalo dobre, pretože táto cesta svetom Delphi bola posledná.

Slovo na záver

Staré príslovie vraví, že všetko sa raz musí skončiť. Výnimkou nie je ani Delfinárium. Počas jedného roka nášho spoločného úsilia sme prešli od základov práce s Delphi cez databázy až po grafiku a multimédiá. Je jasné, že v takomto krátkom seriáli sme si nestihli prebrať všetky možnosti tohto skvelého vývojového nástroja. Nehovorili sme napríklad o tvorbe ActiveX komponentov, OLE Automation klientov a serverov ani o DLL knižniciach. Cieľom seriálu bolo uľahčiť začiatočníkom prácu s Delphi a pokročilejším sprostredkovať niekoľko trikov, o ktorých manuály nepíšu. Pevne dúfam, že Delfinárium splnilo vaše očakávania.

Literatúra:
D. Osier – S. Grobman – S. Batson: Teach yourself Delphi 2 in 21 days.
M. Cantú: Mistrovství v Delphi 2.

■ **Ivan Zernovác ml.**