Typisierte Dateien von Martin Strohal

Einleitung

Herzlich willkommen im Tutorial zum Erstellen und Verwenden sequenzieller Dateien! Als Allererstes: Was sind eigentlich "sequenzielle Dateien" und wozu braucht man sie?

Eine typisierte Datei ist eine Datei, in der eine Folge von immer gleich aufgebauten Datensätzen gespeichert wird. Jeder Datensatz entspricht einem Record-Typ, deshalb "typisierte" Datei.

Jetzt kommt vielleicht der Einwand: Wenn ich über die Datenbankoberfläche eine kleine Paradox- oder dBase-Tabelle anlege, bekomme ich das Gleiche ohne große Schreibarbeit und kann sogar datenbanksensitive Komponenten verwenden. Das ist doch viel einfacher!

Stimmt, es mag einfacher sein, ohne eine Zeile Quellcode auf obige Weise eigene Dateien zu erstellen. Allerdings muss dann auch bei jeder Weitergabe des Programms die Borland Database Engine (BDE) mitgegeben werden. Und die steht mit ca. 8 MB in keinem Verhältnis zu einer "kleinen" Delphi-Anwendung von ein paar Hundert MB.

Verwenden wir dagegen sequenzielle Dateien, brauch nur noch die EXE weitergegeben werden. Dafür ist aber auch etwas Tipparbeit erforderlich. Und was da getippt werden soll, wird im Folgenden erklärt.

Der Record

Als allererstes müssen wir festlegen, wie ein Datensatz aufgebaut sein soll. Als Beispiel wollen wir eine kleine Adressverwaltung erstellen. Die Daten, die in jedem Datensatz gespeichert werden sollen, definieren wir in einem Record:

```
type Adresse = record
       name: string[50];
       strasse: string[100];
       plz: integer;
       ort: string[50];
     end;
```

Da Strings in Delphi 5 dynamisch verwaltet werden, für sie also kein fester Speicherplatz reserviert wird, müssen wir ShortStrings verwenden. Für "Name" ist ein String von 50 Zeichen Länge vorgesehen, somit ist für den Compiler klar, wie viel Speicher für einen kompletten Datensatz reserviert werden muss.

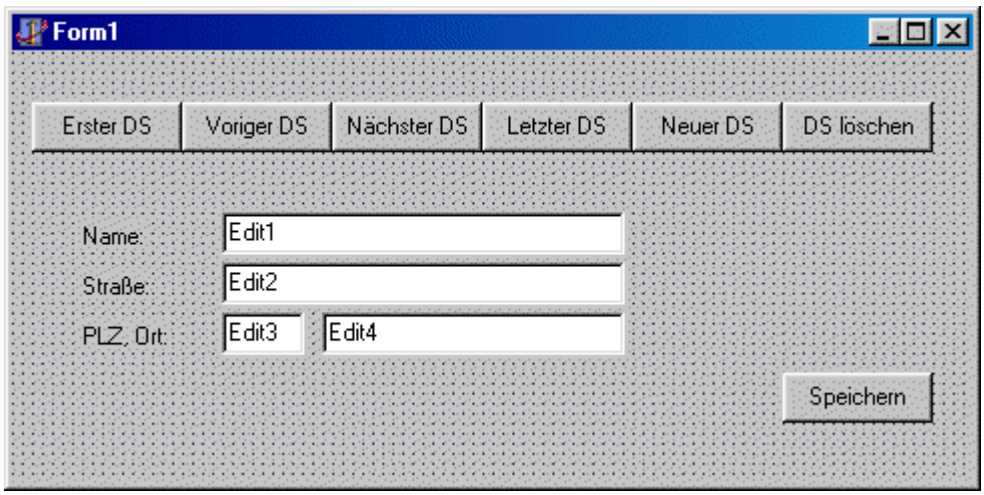
Wichtig: Zwischen den verschiedenen Delphi-Versionen können sich die Speichergrößen der Datentypen ändern. Ein sicherer Zugriff auf eine typisierte Datei ist also nur mit Anwendungen, die von der gleichen Delphi-Version kompiliert wurden, möglich.
Um dies zu umgehen, sollte ein "packed record" und keine generischen Typen wie Integer verwendet werden. Integer steht in Delphi 1 für den Typ SmallInt, ab Delphi 2 für LongInt.

Als nächstes legen wir eine Variable fest, die unsere neu anzulegende Datei "adressen.dat" repräsentieren soll:

```
var datei: file of Adresse;
```
Nun ist also klar, dass die Datei aus einer - theoretisch unendlichen - Folge von Datensätzen des Typs Adresse bestehen soll.

Oberfläche

Bevor wir an die eigentliche Arbeit gehen, basteln wir uns noch eine Oberfläche für die kleine Adressverwaltung. Dazu beginnen wir eine "Neue Anwendung" (Menü Datei) und platzieren vier Eingabefelder (TEdit) auf dem Formular. Mit Labels können die Eingabefelder beschriftet werden (Name, Strasse, Postleitzahl, Ort). Nun brauchen wir noch sechs Buttons, deren Caption wir der Reihe nach wie folgt benennen: Erster Datensatz, voriger Datensatz, nächster Datensatz, letzter Datensatz, neuer Datensatz, Datensatz löschen. Außerdem fehlt noch ein Button, der das Speichern der Eingaben ermöglicht. Nun sollte etwas in folgender Art entstanden sein:



Datei öffnen

Wenn wir unsere Anwendung starten (OnCreate-Ereignis), soll automatisch die Datei "adressen.dat" geöffnet werden. Falls sie noch nicht existiert (beim ersten Start), wird sie neu angelegt:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  AssignFile(datei,'adressen.dat');
  if FileExists('adressen.dat') then ReSet(datei)
  else ReWrite(datei);
end;
```
Assignfile stellt die Verbindung zwischen unserer Datei-Variablen und einem Dateinamen her. Dabei muss die Endung nicht "dat" lauten; es kommt nur darauf an, dass die Datei auch tatsächlich das gleiche Format hat, wie wir die Variable "datei" definiert haben, nämlich vom Typ "Adresse".

Reset öffnet eine existierende Datei; falls sie nicht existiert, tritt ein Fehler auf. *Rewrite* legt eine neue Datei an. Existiert bereits eine gleichen Namens, wird diese überschrieben.

Bevor wir es vergessen, legen wir gleich ein OnClose-Ereignis an, damit die Datei bei Programmende auch wieder ordnungsgemäß geschlossen wird:

```
procedure TForm1.FormDestroy(Sender: TObject);
begin
  CloseFile(datei);
end;
```

Dateizeiger

Bevor es jetzt an die eigentlichen Dateizugriffe geht, ist noch etwas Theoriewissen nötig: Man muss sich eine sequenzielle Datei vorstellen wie eine alte Langspielplatte. Jedes Musikstück darauf repräsentiert dabei einen Datensatz. Und über unser Programm können wir den Lese- (und Schreib-)kopf so positionieren, dass wir den gewünschten Datensatz bearbeiten können. Nach dem Ausführen der Prozedur *Reset* befindet sich der Kopf (auch Dateizeiger genannt) zu Beginn des ersten Datensatzes.

Um die Positionierung des Dateizeigers durchzuführen, wird *Seek* verwendet. Dabei ist zu beachten, dass der erste Datensatz die Nummer 0 erhält. Soll der fünfte Datensatz gelesen werden, muss also folgender Befehl ausgeführt werden:

```
  Seek(datei,4);
```

Ausgelesen werden die Datensätze mit *read*, geschrieben mit *write*. Danach wechseln sie automatisch zum nächsten Datensatz.

Damit man nicht über's Ziel hinausschießt und dadurch eine Fehlermeldung provoziert, gibt es den Befehl *filesize*, der die Anzahl der vorhandenen Records zurückgibt. Und über *eof* (=end of file) kann geprüft werden, ob man sich bereits beim letzten Datensatz befindet. *Filepos* gibt die Nummer des aktuellen Datensatzes zurück (bei Null beginnend).

Jetzt aber wieder zurück zu unserem konkreten Beispiel.

Positionieren

Als erstes legen wir uns eine Prozedur an, die das Positionieren des Dateizeigers vornimmt, überprüft, ob der gewünschte Datensatz überhaupt vorhanden ist und schließlich die gefundenen Daten in unseren Edit-Feldern anzeigt:

```
procedure TForm1.positionieren(datensatz: integer);
var aktdatensatz: Adresse;
begin
  if (datensatz<0) or (datensatz>=FileSize(datei)) then exit
  else begin
    seek(datei, datensatz);
    read(datei, aktdatensatz);
    edit1.text:=aktdatensatz.name;
    edit2.text:=aktdatensatz.strasse;
    edit3.text:=inttostr(aktdatensatz.plz);
    edit4.text:=aktdatensatz.ort;
  end;
end;
```

"Datensatz" enthält dabei die Nummer eines Datensatzes nach Pascal-Zählweise, also mit Null beginnend. Deshalb muss von Filesize(datei) auch eins abgezogen werden, um die (Pascal-)Nummer des letzten Datensatzes zu erhalten.

Da wir die Postleitzahl im Integer-Format speichern, muss die Zahl zur Anzeige in einen String umgewandelt werden.

Navigieren

Nun gehen wir daran, unsere Navigationsbuttons mit Quellcode zu hinterlegen:

```
procedure TForm1.Button1Click(Sender: TObject);  //erster Datensatz
begin
  positionieren(0);
end;

procedure TForm1.Button2Click(Sender: TObject); //voriger Datensatz
begin
  positionieren(FilePos(datei)-2);
  //Nach dem Lesen befindet sich der Datenzeiger HINTER dem aktuellen
Datensatz;
  //deshalb muss um ZWEI nach vorne gegangen werden
end;

procedure TForm1.Button3Click(Sender: TObject); //nächster Datensatz
begin
  positionieren(FilePos(datei));
end;

procedure TForm1.Button4Click(Sender: TObject); //letzter Datensatz
begin
  positionieren(FileSize(datei)-1);
end;
```

Datensatz löschen

Das Löschen eines Datensatzes ist schon etwas schwieriger. Da es dafür keinen eigenen Befehl gibt, müssen wir einen kleinen Umweg gehen: Wir beginnen an der ersten Position hinter dem zu löschenden Datensatz (StartRecord), sofern dort weitere Datensätze folgen, wenn sie also kleiner als die Gesamtzahl (EndRecord) ist. Die Differenz der beiden Werte (Max) ist die Zahl der Datensätze, die zunächst in Buffer kopiert und anschließend jeweils eine Position weiter vorne wieder geschrieben werden. Dabei überschreibt der erste Datensatz den zu löschenden. Der Rest der Datei wird durch Truncate abgeschnitten.

```
const
  BufferSize = 32;

procedure TForm1.Button6Click(Sender: TObject); // Datensatz löschen
var
  Buffer: array[0..BufferSize - 1] of Adresse;  // erhöht Performance
  i, p: Integer;
  StartRecord, EndRecord: Integer;
  Max: Integer;
begin
  // Datenzeiger wird VOR den akt. Datensatz gesetzt
  Seek(Datei, FilePos(Datei)-1);
  // Starten mit dem ersten Datensatz hinter dem zu löschenden
  StartRecord := FilePos(Datei) + 1;
  EndRecord := FileSize(Datei);

  while StartRecord < EndRecord do begin

    // max. BufferSize Datensätze lesen
    Max := BufferSize;
    if Max > EndRecord - StartRecord then
```

```
      Max := EndRecord - StartRecord;
    Seek(Datei, StartRecord);
    for i := 0 to Max - 1 do
      Read(Datei, Buffer[i]);

    // Datensätze zurückschreiben
    Seek(Datei, StartRecord - 1);
    for i := 0 to Max - 1 do
      Write(Datei, Buffer[i]);
  end;

  // Datei ab hier abschneiden
  Truncate(Datei);
  Positionieren(StartRecord-1);
end;
```

Im Beispiel wird die Anzahl der Datensätze durch BufferSize=32 auf 32 beschränkt. Diese Zahl kann natürlich erhöht werden; auf dynamische Arrays sollte jedoch in diesem Tutorial verzichtet werden.

Neuer Datensatz

Zum Anhängen eines neuen Datensatzes wird ähnlich wie beim Löschen vorgegangen: Die Datei wird in ein Array geladen, ein leerer Datensatz wird angefügt und das Ganze anschließend wieder in die Datei gespeichert. Der Dateizeiger wird auf den letzten Datensatz gestellt.

```
const
  BufferSize = 32;

procedure TForm1.Button5Click(Sender: TObject); //neuer Datensatz
var
  Buffer: array[0..BufferSize - 1] of Adresse;
  i, p: integer;
begin
  Seek(datei,0);
  p:=0;
  while not eof(datei) do begin //solange lesen, bis die Datei zu Ende ist
    Read(datei, Buffer[p]);
    inc(p); //entspr. p:=p+1
  end;
  CloseFile(datei); //Datei schließen...
  Buffer[p+1].name:=''; //Am Ende leeren Datensatz anhängen
  Buffer[p+1].strasse:='';
  Buffer[p+1].plz:=0;
  Buffer[p+1].ort:='';
  ReWrite(datei);  //... und neu anlegen
  for i:=0 to p do write(datei, Buffer[i]);
  //Datensätze (bis auf letzten) in Datei schreiben
  positionieren(FileSize(datei)-1);
  //Dateizeiger auf letzten Datensatz positionieren
end;
```

Das Speichern der Eingaben dürfte nun auch kein großes Problem mehr darstellen. Da der Datenzeiger bereits positioniert ist (durch die Anzeige), genügt folgender Code:

```
procedure TForm1.Button7Click(Sender: TObject); //Datensatz speichern
var datensatz: Adresse;
    nr: integer;
begin
  // Datenzeiger VOR den aktuellen Datensatz setzen
  Seek(Datei,FilePos(Datei)-1);
```

```
  nr:=FilePos(datei);
  datensatz.name:=Edit1.Text;
  datensatz.strasse:=Edit2.Text;

  //hier müsste eine Prüfung stattfinden, ob eine Zahl eingegeben wurde!
  //Falls keine gültige Zahl, wird -1 gespeichert
  datensatz.plz:=StrToIntDef(Edit3.Text, -1);

  datensatz.ort:=Edit4.Text;
  write(datei,datensatz);
  positionieren(nr);
end;
```

Da nach dem Write-Befehl der Datenzeiger verschoben wird, wird die Position in der Variablen "nr" abgelegt, um am Ende die richtige Position wiederherzustellen.

Abschluss

Das war auch "schon" alles. Um das komplette Beispiel in Ruhe nachvollziehen zu können, gibt es hier auch den Code als Zip-File (9 KB).
