F I L E S

Verwaltung von Dateien

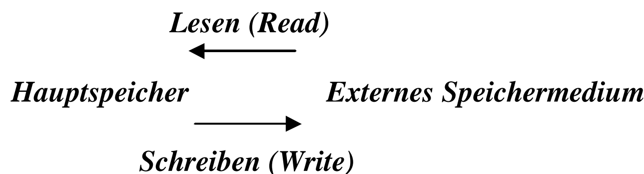
© Herbert Paukert

[01] Laden und Speichern von Dateien (- 02 -)
[02] Der Einsatz von Datenströmen (- 06 -)
[03] Kopieren und Löschen von Dateien (- 08 -)
[04] Ein universeller Byte-Monitor (- 09 -)
[05] Verschlüsselung beliebiger Dateien (- 11 -)
[06] Aufruf externer Fremdprogramme (- 12 -)
[07] Player und Recorder für Sounddateien (- 20 -)
[08] Player für Videodateien (- 27 -)

TYPISIERTE UND UNTYPISIERTE DATEIEN

[01] Laden und Speichern von Dateien (*typdat*)

Eine Datei (File) ist ein Datenbestand (d.h. eigentlich eine Folge von Bytes), welcher auf einem externen Medium abgespeichert ist und auf welchen mit Hilfe eines bestimmten Namens zugegriffen werden kann. Eine solche Datei besteht aus einer Anzahl von Datensätzen (d.h. Teilfolgen von Bytes, die in einer bestimmten Weise strukturiert sind). Je nach Strukturierung dieser Daten (d.h. wie die einzelnen Bytes durch das Programm interpretiert werden), unterscheidet man *Text*-Dateien, *typisierte* Dateien und *untypisierte* Dateien.



Beim Lesen einer Datei werden die Daten von dem externen Medium in den Hauptspeicher transferiert, beim Schreiben erfolgt der Datentransfer in umgekehrter Richtung. Ein interner Puffer dient zur Zwischenspeicherung bei der Ein-/Ausgabe der Daten. Je größer dieser Puffer ausgelegt ist, umso schneller läuft der Datentransfer. Ein interner Positionszeiger enthält stets die jeweils aktuelle Datensatzposition innerhalb der Datei. Prinzipiell erfolgen alle Lese-/Schreiboperationen sequentiell, d.h., nach jedem Lese-/Schreibvorgang wird der interne Positionszeiger automatisch auf den nachfolgenden Datensatz gesetzt.

1. Schritt: Dateitypisierung. Eine Variable ***F*** wird als so genannte Dateivariable deklariert. *var F: TextFile* (Textdatei); *var F: File of Filetyp* (typisierte Datei); *var F: File* (untypisierte Datei). Der Datentyp *Filetyp* beschreibt die Struktur der einzelnen Datensätze (z.B. *Record*).

2. Schritt: Assignation. Diese Prozedur ordnet einer externen Datei über ihren Namen eine Dateivariable zu, sodass die Datei dann im Programmverlauf mit Hilfe dieser Variablen angesprochen werden kann. Die Dateivariable selbst ist intern als Record gespeichert und enthält alle wichtigen Systeminformationen über die Datei (Größe, Name usw.). *AssignFile(F,Dateiname).*

3. Schritt: Eröffnen der Datei. Hier werden interne Routinen veranlasst, welche der Vorbereitung für nachfolgende Lese-/Schreibvorgänge dienen. *Reset(F)* öffnet eine schon bestehende Datei und setzt den internen Positionszeiger auf den Dateianfang. Wird durch die Dateivariable ***F*** eine Textdatei bezeichnet, dann können nur Leseoperationen durchgeführt werden; andernfalls sind sowohl Lese- als auch Schreiboperationen möglich. *Rewrite(F)* erzeugt und eröffnet eine neue Datei. Der Positionszeiger wird auf den Dateianfang gesetzt. Wird durch die Dateivariable ***F*** eine Textdatei bezeichnet, dann können nur Schreiboperationen durchgeführt werden; andernfalls sind sowohl Leseoperationen als auch Schreiboperationen möglich.

4. Schritt: Lesen und Schreiben. Die Prozedur *Read(F,V)* bewirkt bei Textdateien, dass die einzelnen Bytes der Datei mit Hilfe des Positionszeigers nacheinander in einen Zwischenpuffer im Hauptspeicher transferiert werden. Dort werden sie dann entsprechend dem Datentyp der Variablen ***V*** interpretiert (integer, real, char, string) und auf die Variable ***V*** weitertransportiert. Sollen aufeinander folgende Textzeilen (*V: String*) gelesen werden, dann muss die interne Endmarkierung einer Textzeile (*EoLn)* übersprungen und anstelle von *Read* die Anweisung *ReadLn* gesetzt werden.

Die Prozedur *Read(F,V)* bewirkt bei typisierten Dateien, dass jener Datensatz der Datei, auf welchen der interne Positionszeiger verweist, auf die angegebene Variable ***V*** transferiert wird. Die Variable ***V*** und der Datensatz müssen vom gleichen Typ sein.

Nach jedem Lesevorgang wird automatisch der Positionszeiger auf den nachfolgenden Datensatz gestellt. Solange das Dateiende noch nicht erreicht ist, hat die Standardfunktion *Eof(F)* den Wert *False*, ansonsten den Wert *True* (End of File). Die Prozedur *Write(F,V)* schreibt die Daten aus der Speichervariablen ***V*** in die externe Datei ***F***. Es gelten dabei im Wesentlichen die gleichen Regeln wie beim Lesen.

5. Schritt: Fehlerprüfung. DELPHI prüft standardmäßig nach jeder Ein-/Ausgabeoperation auf etwaige Fehler (Datei nicht vorhanden, inkompatible Datentypen usw.) und bricht das Programm gegebenenfalls mit einer Meldung ab. Diese automatische Fehlerprüfung kann mit den speziellen Compilerbefehlen *{$I+}* und *{$I-}* an- und abgeschaltet werden. Bei abgeschalteter Überprüfung bewirkt ein Fehler keinen Programmabbruch. Die Fehlerart lässt sich dann mit Hilfe der Standardfunktion *IOResult* ermitteln. Diese liefert den entsprechenden Fehlercode (beispielsweise bedeutet der Wert Null, dass kein Fehler aufgetreten ist).

6. Schritt: Schließen der Datei. Durch die Prozedur *CloseFile(F)* wird die Verbindung der Dateivariablen ***F*** und der jeweiligen externen Datei gelöst, sodass die Dateivariable zur Assignation anderer Dateien vom gleichen Typ verwendet werden kann.

Text-Dateien

Textdateien bestehen aus unterschiedlich langen Folgen von Zeichen, die man in Zeilen unterteilen kann. Das Ende einer solchen Datenzeile wird durch zwei spezielle Steuerzeichen, *Carriage Return = chr(13)* und *Line Feed = chr(10),* markiert. Dieses Zeilenende kann mit Hilfe der booleschen Standardfunktion *EoLn(F)* abgefragt werden. Das Ende der ganzen Textdatei wird durch das Steuerzeichen *(Ctrl Z = chr(26))* gekennzeichnet und kann mittels *Eof(F)* überprüft werden. Wegen der unterschiedlichen Länge der einzelnen Zeilen kann auf diese nicht direkt zugegriffen werden. Zur Bearbeitung muss daher die ganze Textdatei sequentiell in einen Hauptspeicherbereich (z.B. array of string) eingelesen werden bzw. umgekehrt vom Hauptspeicher sequentiell auf das externe Medium geschrieben werden (sequentieller Dateityp).

Zusätzlich zu den Verwaltungsprozeduren, die oben schon besprochen wurden, gibt es für Textdateien noch weitere spezielle Routinen: *Append* öffnet eine bestehende Textdatei für nachfolgende Schreiboperationen, d.h., neue Datenzeilen werden angehängt. *SetTextBuf* ordnet einer Textdatei einen Zwischenpuffer mit gewünschter Größe zu. *Flush* erzwingt das Ausschreiben des Zwischenpuffers auch dann, wenn er noch nicht ganz gefüllt ist.

Typisierte Dateien

Alle Sätze einer typisierten Datei sind vom gleichen Datentyp (beispielsweise *Record*). Sie sind alle gleich strukturiert und gleich lang. Die Datensätze sind, von Null aufwärts, fortlaufend nummeriert. Die aktuelle Satznummer wird intern im Positionszeiger festgehalten. Auf die Datensätze wird normalerweise in sequentieller Abfolge zugegriffen. Durch *Read* oder *Write* eines Datensatzes wird der Positionszeiger automatisch auf den nachfolgenden Datensatz gestellt. Bei der Datei-Eröffnung zeigt der Positionszeiger immer auf die Position Null.

Bei typisierten Dateien kann auch in direkter Weise auf einen Datensatz zugegriffen werden (Random Access). Die Prozedur *Seek(F,N)* setzt den Positionszeiger auf den N-ten Datensatz der bereits offenen, typisierten Datei ***F***. Nachfolgende *Read(F,V)* oder *Write(F,V)* beziehen sich dann genau auf diesen Datensatz. Dabei ist ***F*** die Dateivariable und ***V*** eine Variable des vorher definierten Datentyps.

Die systemdefinierten Funktionen *FilePos(F)* und *FileSize(F)* bestimmen die aktuelle Satznummer und die aktuelle Dateigröße (gezählt in Datensätzen). Die Prozedur *Truncate(F)* schließlich schneidet die Datei an der momentanen Position *FilePos(F)* ab, wodurch nachfolgende Datensätze entfernt werden.

Untypisierte Dateien

Bei diesem Dateityp macht DELPHI überhaupt keine Annahmen über die Art und Organisation der gespeicherten Daten. Untypisierte Dateien werden ähnlich verwaltet wie typisierte Dateien mit der Ausnahme, dass es keine Typisierung von Datensätzen gibt und dass die Ein-/Ausgabeoperationen blockweise erfolgen. Statt *Read* bzw. *Write* müssen die Befehle *Blockread* bzw. *Blockwrite* gesetzt werden. Untypisierte Dateideklarationen werden vorzugsweise für schnelle Zugriffe auf niedriger Ebene benutzt. Die Zugriffseinheit ist dabei ein Byte.

Dialog-Komponenten zur Dateiauswahl

Das Register *Dialog* der Komponentenpalette stellt zwei Objekte zur bequemen Dateiauswahl zur Verfügung. ***OpenDialog*** zeigt eine Box an, in welcher der Benutzer eine Datei zum Laden auswählen kann. Wird hier eine Datei angegeben, die es im angezeigten Verzeichnis nicht gibt, dann erfolgt eine Fehlermeldung (über die Eigenschaft *OpenDialog.Options = [ofFileMustExist]*). Die Eigenschaften *OpenDialog.DefaultExt* und *OpenDialog.Filter* legen die Erweiterung für den Dateinamen (z.B. *.dat) fest. Existiert die ausgewählte Datei, dann liefert *OpenDialog.FileName* den Dateinamen und die Methode *OpenDialog.Execute* den Wert *True*. Wird hingegen die Auswahlbox abgebrochen (*Escape*), so liefert diese Methode den Wert *False*.

SaveDialog zeigt eine Box an, in welcher der Benutzer eine Datei zum Speichern auswählen kann. Wird hier eine Datei angegeben, die es im angezeigten Verzeichnis schon gibt, dann erfolgt eine Warnmeldung (über die Eigenschaft *SaveDialog.Options = [ofOverwritePrompt]*). Ein Dateiname kann der Box mittels *SaveDialog.FileName* übergeben werden. Die zusätzlichen Eigenschaften *SaveDialog.DefaultExt* und *SaveDialog.Filter* legen die Erweiterung für den Dateinamen fest. Schließt man die Box mit <Okay>, dann liefert die Methode *SaveDialog.Execute* den Wert *True.* Wird hingegen die Auswahlbox abgebrochen (*Escape*), so liefert diese Methode den Wert *False*.

Im den unten stehenden Programmlistings erfolgt die Dateiauswahl mit Hilfe einer Dialogbox und danach wird die ausgewählte Datei gelesen (geladen) bzw. geschrieben (gespeichert). Dabei wird vorausgesetzt, dass *Anz* Datensätze vom Recordtyp *Person* in einem Array abgelegt werden.

```
type  TPerson = record                     // Benutzerdefinierter Datensatz-Typ
        Name    : String[30];              // Namens-Feld
        Telefon : String[30];              // Telefon-Feld
        Geburt  : Integer;                 // Geburtsjahres-Feld
      end;
const Max    = 100;                        // Höchstanzahl der Datensätze

var   Person : TPerson;                    // Ein typisierter Datensatz (record)
      Bereich: array[1..Max] of TPerson;   // Der Datenbereich
      Datei  : File of TPerson;            // Typisierte Dateivariable
      Verz   : String;                     // Verzeichnis-Name
      DName  : String;                     // Datei-Name
      Anz    : Integer;                    // Aktuelle Anzahl der Datensätze

procedure TForm1.Button9Click(Sender: TObject);
{ Datei laden }
begin
  GetDir(0,Verz);                          // bzw. Verz := GetCurrentDir;
  with OpenDialog1 do begin
    InitialDir := Verz;
    Filter     := 'Personaldatei (*.dat)|*.dat';  // Vertical Bar | = [AltGr]+[<]
    DefaultExt := 'dat';
    Options    := [ofFileMustExist];
    FileName   := '';
    if OpenDialog1.Execute then DName := FileName
                           else Exit;
  end;
  AssignFile(Datei,DName);
  {$I-} Reset(Datei);{$I+}
  if IOResult <> 0 then Exit;
```

```
  Anz := 0;
  while not Eof(Datei) do begin
    Anz := Anz + 1;
    Read(Datei,Bereich[Anz]);
  end;
  CloseFile(Datei);
end;

procedure TForm1.Button10Click(Sender: TObject);
{ Datei speichern }
var i : Integer;
begin
  with SaveDialog1 do begin
    InitialDir := Verz;
    Filter     := 'Personaldatei (*.dat)|*.dat';
    DefaultExt := 'dat';
    Options    := [ofOverwritePrompt];
    FileName   := DName;
    if Execute then DName := FileName
               else Exit;
  end;
  AssignFile(Datei,DName);
  {$I-} Rewrite(Datei); {$I+}
  if IOResult <> 0 then Exit;
  For i := 1 to Anz do write(Datei,Bereich[i]);
  CloseFile(Datei);
end;
```

Datenausgabe am Drucker

Die Möglichkeiten der Datenausgabe am Drucker sind vielfältig. In der hier verwendeten Möglichkeit wird der Drucker als Textdatei deklariert (*var PText: TextFile; AssignPrn(PText)*). Dann muss die Dateivariable *PText* zum Schreiben geöffnet werden, sodass die gewünschten Daten ausgedruckt werden können (*Rewrite(PText); Writeln(PText,...)*). Am Ende wird die geöffnete Datei (hier eben der Drucker) geschlossen (*CloseFile(PText)*). Damit das alles funktioniert, muss in der *uses*-Anweisung die Unit *Printers* eingebunden werden. Durch die Verwendung der zusätzlichen Dialogkomponenten *PrintDialog* und *FontDialog* aus dem Register *Dialoge* der Komponentenpalette können bestimmte Druckereinstellungen und Schriftattribute ausgewählt werden.

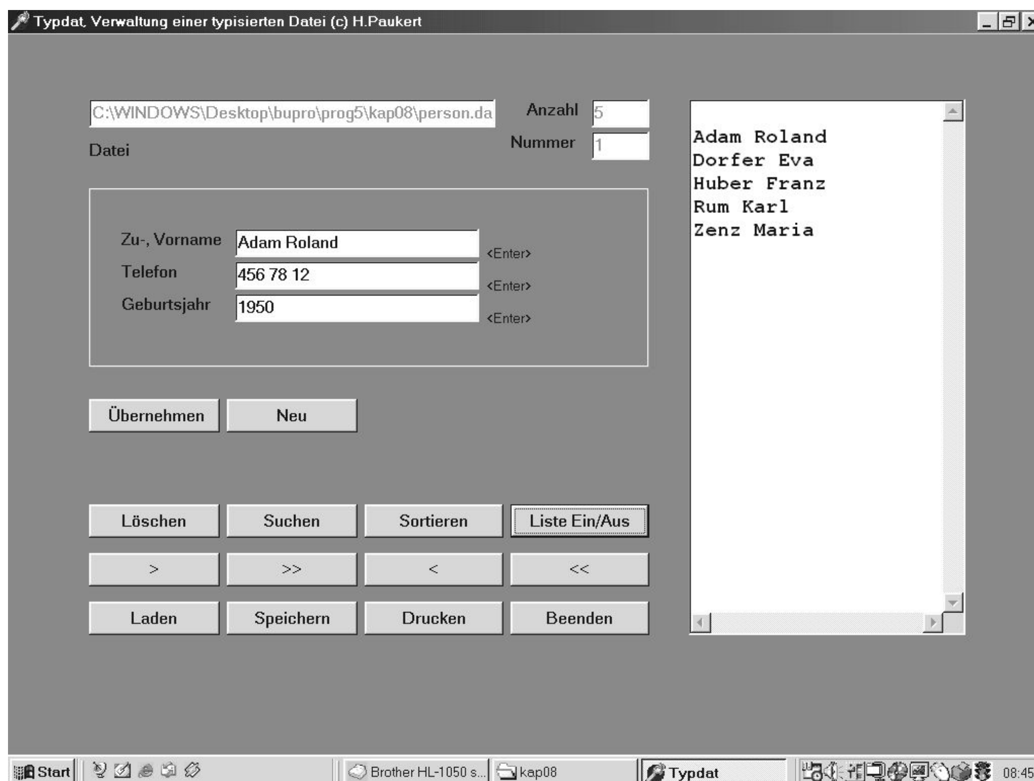
```
procedure TForm1.Button11Click(Sender: TObject);
{ Datei drucken }
var i : Integer;
    PText: TextFile;
begin
  if Anz = 0 then Exit;
  AssignPrn(PText);
  Printer.Canvas.Font.Size := 14;
  Writeln(PText,'------------------------------');
  for i := 1 to Anz do
      with Bereich[i] do
           Writeln(PText,' ',Name,' / ',Telefon);
  Writeln(PText,'------------------------------');
  CloseFile(PText);
end;
```

Das Programmprojekt "typdat"

Dieses Programmprojekt dient der Erzeugung und Verwaltung einer einfachen Datenbank. Jeder Datensatz (record) enthält die Datenfelder (fields) NAME, TELEFON und GEBURTSJAHR. Im Hauptspeicher des Computers wird ein Bereich reserviert, der aus höchstens hundert Datensätzen besteht (array of record). Der Bereich kann auch auf eine externe Datei ausgelagert werden.
Die Variablendefinitionen sind auf der vorangehenden Seite beschrieben und die entsprechenden Verwaltungsroutinen sind auf der nachfolgenden Seite aufgelistet.

Folgende Verwaltungsroutinen können durchgeführt werden:

- Einen neuen Datensatz eingeben und übernehmen (hinzufügen).
- Einen bestehenden Datensatz ändern und übernehmen.
- Den aktuellen Datensatz löschen.
- Einen Datensatz suchen (gesucht wird im Namensfeld).
- Alle Datensätze nach ihren Namensfeldern sortieren und in einer Memobox anzeigen.
- Zum nachfolgenden Datensatz gehen (>).
- Zum letzten Datensatz gehen (>>).
- Zum voranliegenden Datensatz gehen (<).
- Zum ersten Datensatz gehen (<<).
- Eine Datei über eine Auswahlbox in den Hauptspeicher laden.
- Die Datensätze über eine Auswahlbox in eine Datei abspeichern.
- Von allen Datensätzen das Namensfeld und das Telefonfeld ausdrucken.



Das komplette Listing des Programmes "***typdat***" findet der Leser auf der Begleit-CD.

[02] Der Einsatz von Datenströmen (*stream*)

Der Klassentyp *TStream* besitzt mächtige Methoden zum Lesen und Schreiben von Datenobjekten. Das Besondere dabei ist, dass diese Objekte verschiedene Struktur aufweisen können. In einen Stream können also beispielsweise Daten vom Typ eines Records, numerische Daten, Textdaten, usw. vorkommen.

Im Programm "***stream***" werden verschiedene Datenobjekte hintereinander in einen so genannten *FileStream* verpackt und in eine externen Datei ausgelagert bzw. aus dieser Datei wieder in den Hauptspeicher geladen. Diese Technik eignet sich u.a. zur temporären Sicherung der vielfältigen Datenobjekte eines Formulars - bei Bedarf können sie dann wieder abgerufen werden.

Einige wichtige Eigenschaften und Methoden von *TFileStream* sollen kurz erklärt werden:

```
var fStream: TFileStream;                          // Deklaration einer Instanz
    fName: String;                                 // Name der entsprechenden Datei

fStream := TFileStream.Create(fName,fmCreate);     // Erzeugung eines neuen Streams
fStream := TFileStream.Create(fName,fmOpenRead);   // Öffnen eines Streams zum Lesen
fStream := TFileStream.Create(fName,fmOpenWrite);  // Öffnen eines Streams zum Schreiben
fStream.ReadBuffer(Datenobjekt,Datengröße);        // Lesen eines Datenobjektes
                                                   // (die Größe liefert die Funktion SizeOf)
fStream.WriteBuffer(Datenobjekt,Datengröße);       // Schreiben eines Datenobjektes
fStream.Position;                                  // Positioniert den internen Dateizeiger
fStream.Free;                                      // Entfernt den Stream aus dem Speicher
```

Wie aus dem Quellcode ersichtlich ist, werden die verschiedenen Datenobjekte im FileStream sequentiell gelesen oder geschrieben. Interessant ist die Handhabung von Zeichenketten (Strings). Zuerst muss die Stringlänge geschrieben werden und dann genügt die Angabe des ersten Zeichens. Beim Lesen muss zuerst die Stringlänge gelesen, dann mit dieser Länge der String genau begrenzt (*SetLength*) und schließlich der String zeichenweise gelesen werden.

```
const fName0  = 'data.stm';

type  TPerson = record
         Name   : String[40];
         Geburt : TDate;
      end;

var   Person : TPerson;
      Geld   : Real;
      Info   : String;
      fStream: TFileStream;
      fName  : String;

procedure WriteStream;
// Die Daten sequentiell in den FileStream schreiben
var Len : Integer;
begin
  fName := GetCurrentDir + '\' + fName0;
  fStream := TFileStream.Create(fName,fmCreate);
  with fStream do begin
    Position := 0;
    WriteBuffer(Person,SizeOf(Person));
    WriteBuffer(Geld,SizeOf(Geld));
    Len := Length(Info);
    WriteBuffer(Len,SizeOf(Len));
    WriteBuffer(Info[1],Len);
  end;
  fStream.Free;
end;

procedure ReadStream;
// Den FileStream sequentiell in die Daten lesen
var Len : Integer;
begin
  fName := GetCurrentDir + '\' + fName0;
  fStream := TFileStream.Create(fName0,fmOpenRead);
  with fStream do begin
    Position := 0;
    ReadBuffer(Person,SizeOf(Person));
    ReadBuffer(Geld,SizeOf(Geld));
    ReadBuffer(Len,SizeOf(Len));
    SetLength(Info,Len);
    ReadBuffer(Info[1],Len);
  end;
  fStream.Free;
end;
```

[03] Kopieren und Löschen von Dateien (*fileman*)

Zu den wichtigsten Verwaltungsroutinen mit Dateien gehören das **Kopieren** und das **Löschen**. Diese Routinen lassen sich in verschiedener Weise programmieren. Die wohl eleganteste Methode ist jene mit Hilfe der **Stream**-Technik.

Dazu wird erstens eine Instanz ***s1*** eines FileStreams erzeugt, welche zum Lesen der Daten einer Quelldatei (*Name0*) dient. Zweitens wird eine Instanz ***s2*** eines FileStreams erzeugt, welche zum Schreiben der Daten in eine Zieldatei (*Name1*) dient. Mit der Methode ***CopyFrom*** werden die Daten aus dem ersten FileStream in den zweiten FileStream kopiert. Damit ist dann alles erledigt.

Zum Löschen einer Datei (*Name*) wird die im System vordefinierte boolesche Funktion ***DeleteFile*** verwendet. Sie liefert den Wert *True*, wenn alles fehlerlos funktioniert hat, ansonsten den Wert *False*. Außerdem wird noch die vordefinierte boolesche Funktion ***FileExists****(Name)* eingesetzt, welche überprüft, ob eine Datei mit dem Dateinamen *Name* überhaupt auf dem aktuellen Datenträger existiert.

Der Dateiname *Name* muss dabei natürlich eine vollständige Pfadangabe enthalten. Will man aus einem vollständigen Dateinamen nur den eigentlichen Namen gewinnen, so kann man dazu die Funktion *ExtractFileName* verwenden.

```
procedure FileCopy(Name0,Name1: String);
// Kopiert die Datei "Name0" auf die Datei "Name1"
var s1,s2: TFileStream;
    t: String;
    x: Integer;
begin
  t := ' Quelldatei "' + ExtractFileName(Name0) + '" nicht gefunden!';
  if NOT FileExists(Name0) then begin
     ShowMessage(t);
     Exit;
  end;
  t := ' Zieldatei "' + ExtractFileName(Name1) + '" überschreiben?';
  if FileExists(Name1) then begin
     x := MessageBox(0,PChar(t),'Frage',36);
     if x = 7 then Exit;
  end;
  try
    s1 := TFileStream.Create(Name0,fmOpenRead);
    s2 := TFileStream.Create(Name1,fmCreate);
    s2.CopyFrom(s1,s1.Size);
    s1.Free;
    s2.Free;
  except
    ShowMessage('KEINE Datei kopiert!');
    Exit;
  end;
  ShowMessage('Datei erfolgreich kopiert!');
end;

procedure FileDelete(Name: String);
// Löscht die Datei "Name"
var Res : Boolean;
begin
  if NOT FileExists(Name) then begin
     ShowMessage(' Datei "' + Name + '" nicht gefunden!');
     Exit;
  end;
  Res := DeleteFile(Name);
  if NOT Res then ShowMessage(' Datei NICHT gelöscht!')
             else ShowMessage(' Datei erfolgreich gelöscht!');
end;
```

[04] Ein universeller Byte-Monitor (x*view*)

Im untypisierten Dateiformat wird eine Datei als unstrukturierter Fluss von Bytes angesehen. Im Textformat besteht eine Datei aus Zeichenketten (Strings) und im typisierten Format sind die einzelnen Datensätze strukturierte Record-Typen.

Bei der Verwendung des untypisierten Dateiformates wird durch die Befehle ***Reset(F,1)*** und ***BlockRead(F,Puffer,PufLen,Resr)*** die Datei mit der Dateivariablen ***F*** geöffnet und byteweise in den Puffer-Bereich (*Array[1..PufLen] of Byte*) eingelesen. Die Steuervariable ***Resr*** gibt die echte Anzahl der gelesenen Bytes an. Offensichtlich gilt *Resr = PufLen*, solange das physische Ende der Datei nicht erreicht ist. Wenn das aber der Fall ist, dann muss *Resr < PufLen* sein. Die Befehle ***Blockwrite*** und ***Rewrite*** bilden das genaue Gegenteil zu ***Blockread*** und ***Reset***. Sie schreiben die Bytes aus dem Puffer auf die Datei. Der Befehl *Filepos(F)* liefert die echte Nummer des Datensatzes (hier genau EIN Byte), auf welchen gerade zugegriffen wird. Die Nummerierung beginnt dabei bei Null. Der Befehl *Seek(F,N)* positioniert den internen Datensatzzeiger auf den N-ten Datensatz, sodass auf ihn zugegriffen werden kann (hier genau auf das N-te Byte).

Im Programm "***xview***" wird eine beliebige Datei in ein ***dynamisches Array of Byte*** blockweise eingelesen. Die entsprechende Routine *LoadFile* liefert die Dateilänge zurück. Sodann werden die einzelnen Bytes mit Hilfe der Prozedur *ByteToChar* in eine RichEdit-Komponente übertragen. In der zusätzlichen Prozedur *SaveFile* können die Bytes des dynamischen Speicher-Arrays wieder blockweise in eine Datei zurückgeschrieben werden.

Was ist ein ***dynamisches Array***? Im Gegensatz zu den statischen Arrays wird bei dynamischen Arrays bei der Deklaration die Größe nicht festgelegt. Stattdessen gibt es eine Funktion *SetLength*, mit der zur Laufzeit die Arraygröße vorläufig festgelegt wird. Erst dann kann auf die Elemente des Arrays über einen entsprechenden Index zugegriffen werden, wobei der erste Index immer 0 ist. Bei Bedarf ist es möglich, die Arraygröße zu erweitern oder auch zu verkleinern.

```
var A: Array of Real;
    i,n: Integer;

begin
  SetLength(A,10);
  for i := Low(A) to High(A) do A[i] := Sqrt(i);
  SetLength(A,15);
  for i := 10 to 14 do A[i] := Sqrt(i);
  A := Copy(A,0,10);
  n := Length(A);
  A := NIL;
end;
```

Obiges Beispiel zeigt, dass es noch zusätzliche Standard-Funktionen für dynamische Arrays gibt: die *Copy*-Funktion reduziert die Arraygröße, die Zuweisung von *NIL* bewirkt eine Freigebung des ganzen Arrayspeichers, *Length(A)* liefert die aktuelle Anzahl der Array-Elemente, *Low(A)* ist der niedrigste Indexwert (also immer 0) und *High(A)* der höchste Indexwert. Falls das Array noch keine Elemente hat, dann liefert *High(A)* den Wert -1.

```
type Feld = Array of Byte;

var  FByte: Feld;
     Verz,FName: String;
     FLen: Integer;

procedure ByteToChar(RE: TRichEdit; var FByte: Feld; N: Integer);
// Transfer der Bytes als Characters nach RichEdit
var S: String;
    C: Char;
    B: Byte;
    I: Integer;
```

```
begin
  RE.Clear;
  S := '';
  for I := 0 to N do begin
    B := FByte[I];
    C := '.';
    if (B > 31) or (B = 10) or (B = 13) then C := Chr(FByte[I]);
    S := S + C;
  end;
  RE.Text := S;
end;

function LoadFile(FName: String; var FByte: Feld): Integer;
// Routine zum byteweisen Laden einer beliebigen Datei FName
// in ein dynamisches Byte-Array FByte

const FL = 16284;
var   Puffer: Array [1..FL] of Byte;   // Pufferspeicher
      F     : File;                    // Untypisierte Datei
      L,N,I,Resr: Integer;             // Hilfsvariable

begin
  AssignFile(F,FName);                 // Zuordnen einer Dateivariablen
  Reset(F,1);                          // Öffnen der untypisierten Datei
  Result := FileSize(F);
  N := 0;
  SetLength(FByte,1);

  Repeat                               // Wiederholen, bis Dateiende erreicht
    L := Length(FByte);
    SetLength(FByte,L+FL);
    BlockRead(F,Puffer,FL,Resr);       // Blockweises Lesen der Bytes
    if Resr > 0 then begin             // Wenn Dateiende nicht erreicht, dann
       for I := 1 to Resr do begin     // Übertragen der Bytes nach FByte
           FByte[N] := Puffer[I];
           N := N + 1;
       end;
    end;
  until (Resr < FL);                   // Ende der Wiederholung bei Dateiende

  CloseFile(F);                        // Schließen der Datei
end;

procedure SaveFile(FName: String; var FByte: Feld; N: Integer);
// Routine zum byteweisen Speichern einer beliebigen Datei FName
// aus einem dynamischen Byte-Array FByte

const FL = 1024;
var   Puffer: Array [1..FL] of Byte;  // Pufferspeicher
      F     : File;                   // Untypisierte Datei
      I,Resr,Resw : Integer ;         // Hilfsvariable

begin
  AssignFile(F,FName);                 // Zuordnen einer Dateivariablen
  Rewrite(F,1);                        // Öffnen der untypisierten Datei
  Resr := 0;

  For I := 0 to N-1 do begin           // Wiederholen, bis Dateiende erreicht
    Resr := Resr + 1;
    Puffer[Resr] := FByte[I];
    if (Resr = FL) or (I = N-1) then begin
       BlockWrite(F,Puffer,Resr,Resw); // Blockweises Schreiben der Bytes
       Resr := 0;
    end;
  end;                                 // Ende der Wiederholung

  CloseFile(F);                        // Schließen der Datei
end;
```

[05] Verschlüsselung beliebiger Dateien (*xcode*)

Das Programm "***xcode***" liest eine beliebige Datei byteweise in ein statisches Hauptspeicherarray. Dort werden dann die einzelnen Bytes mit einem Schlüselwort codiert und zurück auf die Datei geschrieben. Dadurch wird die Information verschlüsselt und somit unleserlich. Eine neuerliche Programmdurchführung mit demselben Schlüsselwort transformiert die Daten wieder in ihre ursprüngliche Form.

Zuerst wird das Schlüsselwort *Wort* verdeckt in einem Editfeld eingegeben. Dazu wird die Eigenschaft *TEdit.PasswordChar* auf '***** ' gesetzt. Dann wird die Datei mit dem Namen *Name* mit Hilfe der Routine ***XORCode*** verschlüsselt. In dieser Routine wird die Datei blockweise in den Speicherarray gelesen. Dort werden die einzelnen Bytes **erstens** mit den Buchstaben des Schlüsselwortes, **zweitens** mit der jeweiligen Dateiposition und **drittens** mit der Dateilänge verschlüsselt. Die Verschlüsselung erfolgt mit Hilfe der logischen XOR-Operation. Dann werden die Bytes wieder in die Datei an die richtige Position zurückgeschrieben und der nächste Byteblock wird geladen, codiert und zurückgeschrieben. Dieses Verfahren wird solange wiederholt, bis alle Bytes der Datei verbraucht sind.

```
procedure XORCode(Name: String; Wort: ShortString);
// Routine zur XOR-Verschlüsselung einer beliebigen Datei
// Name = Dateinamen, Wort = Passwort
const FL = 8142;
var   Feld : Array [1..FL] of Byte;    // Pufferspeicher
      F    : File of Byte;             // Dateivariable
      K,I,Resr,Resw: Integer;          // Hilfsvariable
      Posi,L       : Integer;          // Byteposition, Dateilänge
begin
  AssignFile(F,Name);                  // Zuordnen einer Dateivariablen
  Reset(F);                            // Öffnen der untypisierten Datei
  L := FileSize(F);                    // Ermitteln der Dateilänge
  Repeat                               // Wiederholen, bis Dateiende erreicht
    BlockRead(F,Feld,FL,Resr);         // Blockweises Lesen der Bytes
    Posi := FilePos(F);                // Ermitteln der aktuellen Byteposition
    if Resr > 0 then begin             // Wenn Dateiende nicht erreicht, dann
       for I := 1 to Resr do begin     // XOR-Codierung der Bytes des Blockes
           K := (I mod Length(Wort));  // Zeichen des Passwortes bestimmen
           Feld[I] := Feld[I] xor Ord(Wort[K]);    // Byte XOR Passwortzeichen
           Feld[I] := Feld[I] xor (I and 255);     // Byte XOR Byteposition
           Feld[I] := Feld[I] xor (L and 255);     // Byte XOR Dateilänge
       end;
    end;
    Seek(F,Posi - Resr);               // Dateizeiger zurückpositionieren
    BlockWrite(F,Feld,Resr,Resw);      // Schreiben der codierten Byteblöcke
  until (Resr < FL);                   // Ende der Wiederholung bei Dateiende
  Close(F);                            // Schließen der Datei
end;
```

Wie funktioniert die XOR-Operation? Mit dieser Operation werden zwei Bits a und b miteinander zu einem neuen Bit c verknüpft und zwar entsprechend dem logischen "entweder - oder", d.h., die Operation liefert nur dann den Wert 1, wenn die beiden gegebenen Bits verschiedene Werte aufweisen:

Bit a	Bit b	Bit c = a XOR b
0	0	0
0	1	1
1	0	1
1	1	0

In DELPHI bezieht sich der XOR-Operator immer auf zwei Bytes, deren acht Bits gemäß obiger Tabelle miteinander verknüpft werden. Es sei nun w ein bestimmtes Byte und wir verknüpfen ein beliebiges Byte x mit w, sodass z := x XOR w ist. Verknüpft man nun das Ergebnis z noch einmal mit dem Byte w, dann erhält man wieder das Byte x, d.h., ((x XOR w) XOR w) ergibt wieder x. Diesen Sachverhalt verwendet man zur Verschlüsselung und Entschlüsselung, wobei das Byte w die Rolle eines Passwortes übernimmt. Im nachfolgenden Beispiel sei w der ANSI-Code von 'A', also w = ord('A') = 65 (0100 0001). Damit wollen wir beispielsweise den Buchstaben 'E', also x = ord('E') = 69 (0100 0101), verschlüsseln und auch wieder entschlüsseln.

1. Schritt: Verschlüsseln.

```
x              = 0100 0101 ('E')       // Originales Byte
w              = 0100 0001 ('A')       // Passwort
-------------------------------
z = (x XOR w) = 0000 0100              // Verschlüsseltes Byte
```

2. Schritt: Entschlüsseln.

```
z              = 0000 0100             // Verschlüsseltes Byte
w              = 0100 0001 ('A')       // Passwort
-------------------------------
y = (z XOR w) = 0100 0101 ('E')        // Originales Byte
```

Als Ergebnis erhalten wir wieder den Buchstaben 'E'.

[06] Aufruf externer Fremdprogramme (*exec*)

Das Programm "***exec***" soll den Aufruf von Fremdprogrammen und auch deren Beendigung demonstrieren. Das Programm eignet sich besonders gut, um Multimedia-Dateien (Audios und Videos) aus einem DELPHI-Programm über den WINDOWS-MediaPlayer oder ein anderes Multimedia-Programm als parallele Tasks abzuspielen.

Das Windows Application Programming Interface (WinAPI)

WinAPI ist die Schnittstelle für Windows-Anwendungsprogramme. Mit ihrer Hilfe können vom Programmierer in höheren Programmiersprachen wie C oder DELPHI über eigene Programmroutinen die Dienstprogramme des Betriebssystems aufgerufen werden. Diese Programmroutinen sind in dynamischen Dateibibliotheken (DLLs) gespeichert, beispielsweise in ***kernel32.dll***, die bereits im Betriebssystem integriert sind. Die nachfolgenden Funktionen greifen tief in das Task-Management von WINDOWS ein und sollen dem Leser einen Eindruck über systemnahes Programmieren vermitteln.

Ein praktisches Anwendungsbeispiel dazu: Ein Foto ist als eine Grafikdatei „bild.jpg" in einem Windows-Ordner abgespeichert. Mit einem doppelten Mausklick auf den Dateinamen wird automatisch ein vorhandenes Grafikprogramm aufgerufen, mit dessen Hilfe das Foto am Bildschirm dargestellt wird. Voraussetzung dafür ist, dass den Grafikdateien mit der Extension „jpg" das Grafikprogramm, beispielsweise „paint.exe", zugeordnet (registriert) wurde. Die Grafikdatei nennt man dann „Client" und das zugeordnete Grafikprogramm heißt „Server".

Mit der ersten Funktion ***ExecuteFile*** kann beispielsweise aus einem DELPHI-Programm das zu einer Grafikdatei (Client) registrierte Grafikprogramm (Server) aufgerufen und die Grafikdatei geöffnet werden. Die zweite Funktion ***GetExeForFile*** ermittelt das Serverprogramm zu einer Clientdatei. Die dritte Funktion ***KillExe*** beendet ein laufendes EXE-Programm.

function ExecuteFile(FileName, Params, DefaultDir: string; ShowCmd: Integer): THandle;

```
// Eine Datei oder ein Programm "FileName" ausführen
var zFileName, zParams, zDir: array[0..79] of Char;
begin
  Result := ShellExecute(Application.MainForm.Handle,
                         nil,
                         StrPCopy(zFileName, FileName),
                         StrPCopy(zParams, Params),
                         StrPCopy(zDir, DefaultDir),
                         ShowCmd);
end;
```

Diese Funktion dient der Ausführung externer Files und stützt sich auf die interne Delphi-Routine ***ShellExecute***, die in der Unit ***ShellApi*** gespeichert ist. Dabei haben die übergebenen Parameter folgende Bedeutungen:

(1) *FileName*: Name der Datei. Falls es sich dabei um ein ausführbares Programm (EXE-File) handelt, so wird dieses direkt ausgeführt (wenn vorhanden). Falls es sich dabei um eine im Betriebssystem zu einem Server-Programm registrierte Client-Datei handelt, so wird automatisch das Server-Programm ausgeführt, welches dann auf die Client-Datei zugreift.

(2) *Params*: Optionale Angabe von Parametern, die verwendet werden sollen.

(3) *DefaultDir*: Optionale Angabe des Verzeichnisses, auf das zugegriffen werden soll.

(4) *ShowCmd*: Wichtiger Parameter, welcher das Fensterformat des ausgeführten Programmes bestimmt, z.B. *sw_ShowNormal*: Fenster in normaler Größe,
sw_ShowMinimized: Fenster in minimaler Größe,
sw_ShowMaximized: Fenster in maximaler Größe.

Der Rückgabewert der Routine ist eine ganze Zahl, welche bei einer erfolgreichen Programmausführung die Nummer der Applikation (Handle) oder andernfalls einen Fehlercode darstellt.

function GetExeForFile(FileName: String): String;

```
// Zu einer Datendatei (Client)
// das registrierte Windows-Programm (Server) ermitteln
var x: Integer;
begin
  SetLength(Result, MAX_PATH);
  if FindExecutable(PChar(FileName), nil, PChar(Result)) >= 32 then
     SetLength(Result, StrLen(PChar(Result)))
  else Result := IntTostr(x);
end;
```

function KillExe(ExeName: string): integer;

```
// Ein laufendes EXE-Programm entfernen mittels "ProgrammNamen"
const PROCESS_TERMINATE = $0001;
var   ContinueLoop: BOOL;
      FSnapshotHandle: THandle;
      FProcessEntry32: TProcessEntry32;
begin
  result := 0;
  FSnapshotHandle := CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
  FProcessEntry32.dwSize := Sizeof(FProcessEntry32);
  ContinueLoop := Process32First(FSnapshotHandle,FProcessEntry32);
  while integer(ContinueLoop) <> 0 do begin
    if ((UpperCase(ExtractName(FProcessEntry32.szExeFile)) = UpperCase(ExeName))
    or (UpperCase(FProcessEntry32.szExeFile) = UpperCase(ExeName))) then
      Result := Integer(TerminateProcess(OpenProcess(PROCESS_TERMINATE, BOOL(0),
                                   FProcessEntry32.th32ProcessID), 0));
    ContinueLoop := Process32Next(FSnapshotHandle,FProcessEntry32);
  end;
  CloseHandle(FSnapshotHandle);
end;
```

function KillTask(ExeName: string): integer;

```
// EXE-Programm beenden
var PName: String;
begin
  PName := Trim(ExtractFileName(ExeName));
  if PName <> '' then begin
     KillExe(PName);
  end;
end;
```

function KillServProg(FileName: String): Integer;

```
// Clientdatei mithilfe des Serverprogramms beenden
var S: String;
begin
  S := GetExeForFile(FileName);
  KillTask(S);
end;
```

Nachfolgend wird der Quellcode des gesamten Programmes „***exec.exe***“ aufgelistet.
Im Programm ist die zusätzliche Unit „***paustop_u***“ eingefügt. Diese Unit verhindert, dass ein bereits im Speicher laufendes Programm ein zweites Mal ausgeführt wird.
Den Quellcode der Unit findet man am Ende des Listings.

unit exec_u;

```
// Ausführung externer Dateien (c) H.Paukert

interface
uses Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
     Dialogs, StdCtrls, ShellAPI, TLHelp32, paustop_u;
type
  TForm1 = class(TForm)
    OpenDialog1: TOpenDialog;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Button5: TButton;
    Button6: TButton;
    Button7: TButton;
    procedure FormActivate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure Button6Click(Sender: TObject);
    procedure Button7Click(Sender: TObject);
  private { Private-Deklarationen }
  public  { Public-Deklarationen }
  end;

var Form1: TForm1;

implementation
{$R *.DFM}

var Verz : String;
    FName: String;
    Ext  : String;
```

function ExecuteFile(FileName, Params, DefaultDir: string;

```
                   ShowCmd: Integer): THandle;
// Führt ein Programm oder eine Datei aus
var zFileName, zParams, zDir: array[0..79] of Char;
begin
  Result := ShellExecute(Application.MainForm.Handle,
                         nil,
                         StrPCopy(zFileName, FileName),
                         StrPCopy(zParams, Params),
                         StrPCopy(zDir, DefaultDir),
                         ShowCmd);
end;
```

function GetExeForFile(FileName: String): String;

```
// Ermittelung des Serverprogramms zu einer Clientdatei
var x: Integer;
begin
  SetLength(Result, MAX_PATH);
  if FindExecutable(PChar(FileName), nil, PChar(Result)) >= 32 then
     SetLength(Result, StrLen(PChar(Result)))
  else Result := inttostr(x);
end;
```

function KillExe(ExeName: string): integer;
```
 // Ein laufendes EXE-Programm entfernen mittels "ProgrammNamen"
const PROCESS_TERMINATE = $0001;
var   ContinueLoop: BOOL;
      FSnapshotHandle: THandle;
      FProcessEntry32: TProcessEntry32;
begin
  result := 0;
  FSnapshotHandle := CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
  FProcessEntry32.dwSize := Sizeof(FProcessEntry32);
  ContinueLoop := Process32First(FSnapshotHandle,FProcessEntry32);
  while integer(ContinueLoop) <> 0 do begin
    if ((UpperCase(ExtractName(FProcessEntry32.szExeFile)) = UpperCase(ExeName))
    or (UpperCase(FProcessEntry32.szExeFile) = UpperCase(ExeName))) then
      Result := Integer(TerminateProcess(OpenProcess(PROCESS_TERMINATE, BOOL(0),
                                   FProcessEntry32.th32ProcessID), 0));
    ContinueLoop := Process32Next(FSnapshotHandle,FProcessEntry32);
  end;
  CloseHandle(FSnapshotHandle);
end;
```

function KillTask(ExeName: string): integer;
```
// EXE-Programm beenden
var PName: String;
begin
  PName := Trim(ExtractFileName(ExeName));
  if PName <> '' then begin
     KillExe(PName);
  end;
end;
```

function KillServProg(FileName: String): Integer;
```
// Clientdatei mithilfe des Serverprogramms beenden
var S: String;
begin
  S := GetExeForFile(FileName);
  KillTask(S);
end;
```

procedure TForm1.FormActivate(Sender: TObject);
```
// Initialisierungen
begin
  Form1.Width  := Button6.Left + Button7.Width + 50;
  Form1.Height := Button7.Top + 3 * Button6.Height;
  GetDir(0,Verz);
end;
```

procedure TForm1.Button1Click(Sender: TObject);
```
// Eine EXE-Datei ausführen
var S: String;
    H: THandle;
begin
  With OpenDialog1 do begin
       InitialDir := Verz;
       Filter := 'EXE-Dateien (*.exe)|*.exe';
       DefaultExt := '.exe';
       Options    := [ofFileMustExist];
       FileName   := '';
```

```
        if Execute then begin
           S := FileName; FName := S;
           try
             GetDir(0,Verz);
             H := ExecuteFile(S,'',Verz,SW_SHOWNORMAL);
           except
             ShowMessage('Dateifehler !');
           end;
        end;
   end;
end;
```

procedure TForm1.Button2Click(Sender: TObject);

```
// PDF-Dateien auswählen und ausführen
var S: String;
    H: THandle;
begin
  With OpenDialog1 do begin
        InitialDir := Verz;
        Filter := 'PPS-Dateien (*.pps)|*.pps|' +
                  'PPT-Dateien (*.ppt)|*.ppt';
        DefaultExt := '.pps';
        Options    := [ofFileMustExist];
        FileName   := '';
        if Execute then begin
           S := FileName; FName := S;
           try
             GetDir(0,Verz);
             H := ExecuteFile(S,'',Verz,SW_SHOWNORMAL);
           except
             ShowMessage('Dateifehler !');
           end;
        end;
   end;
end;
```

procedure TForm1.Button3Click(Sender: TObject);

```
// PDF-Dateien auswählen und ausführen
var S: String;
    H: THandle;
begin
  With OpenDialog1 do begin
        InitialDir := Verz;
        Filter := 'PDF-Dateien (*.pdf)|*.pdf';
        DefaultExt := '.pdf';
        Options    := [ofFileMustExist];
        FileName   := '';
        if Execute then begin
           S := FileName; FName := S;
           try
             GetDir(0,Verz);
             H := ExecuteFile(S,'',Verz,SW_SHOWNORMAL);
           except
             ShowMessage('Dateifehler !');
           end;
        end;
   end;
end;
```

procedure TForm1.Button4Click(Sender: TObject);

```
//  Audio-Dateien auswählen und abspielen
var S: String;
    H: THandle;
begin
  With OpenDialog1 do begin
       InitialDir := Verz;
       Filter := 'WAV-Dateien (*.wav)|*.wav|' +
                 'MP3-Dateien (*.mp3)|*.mp3|' +
                 'Alle Dateien (*.*) |*.*';
       DefaultExt := '.wav';
       Options    := [ofFileMustExist];
       FileName   := '';
       if Execute then begin
          S := FileName; FName := S;
          try
            GetDir(0,Verz);
            H := ExecuteFile(S,'',Verz,SW_SHOWNORMAL);
          except
           ShowMessage('Dateifehler !');
          end;
       end;
   end;
end;
```

procedure TForm1.Button5Click(Sender: TObject);

```
// Videodateien auswählen abspielen
var S: String;
    H: THandle;
begin
  With OpenDialog1 do begin
       InitialDir := Verz;
       Filter := 'WMV-Dateien (*.wmv)|*.wmv|' +
                 'MPG-Dateien (*.mpg)|*.mpg|' +
                 'MP4-Dateien (*.mp4)|*.mp4|' +
                 'Alle Dateien (*.*) |*.*';
       DefaultExt := '.wmv';
       Options    := [ofFileMustExist];
       FileName   := '';
       if Execute then begin
          S := FileName; FName := S;
          try
            GetDir(0,Verz);
            H := ExecuteFile(S,'',Verz,SW_SHOWNORMAL);
          except
            ShowMessage('Dateifehler !');
          end;
       end;
   end;
end;
```

procedure TForm1.Button6Click(Sender: TObject);

```
// Laufenden Prozess beenden
begin
  Ext := ExtractFileExt(FName);
  if Ext = '.exe' then KillTask(FName)
                  else KillServProg(FName);
end;
```

procedure TForm1.Button7Click(Sender: TObject);

```
// Programmende
begin
  Application.Terminate;
end;

end.
```

Die nachfolgende Unit verhindert, dass ein bereits im Speicher laufendes Programm ein zweites Mal ausgeführt wird.

unit paustop_u;

```
// verhindert mehrfachen Aufruf eines Programmes

interface

implementation
uses forms, windows;
var  mutex : THandle;
     h : HWnd;

initialization
  mutex := CreateMutex(nil,true,'MyXYZMutex');
  if getLastError = ERROR_ALREADY_EXISTS then begin
     h := 0;
     repeat
       h := FindWindowEx(0,h,'TApplication',PChar(Application.Title))
     until h <> application.handle;
     if h <> 0 then begin
        Windows.ShowWindow(h, SW_ShowNormal);
        windows.SetForegroundWindow(h);
     end;
     halt;
  end;

finalization
  ReleaseMutex(mutex);
end.
```

[07] Player und Recorder für Sounddateien (*pausound*)

Das Programm "***pausound***" dient dem Abspielen und dem Aufnehmen von Sound-Dateien im WAVE-Format (wav).

```
unit pausound_u;
// Soundrecorder und SoundPlayer für WAVE-Dateien (c) H. Paukert
{ ==============================================================================
Allgemeine Hinweise zur Multimedia-Programmierung:

(1) Die Verwendung der Multi-Media-Komponente "TMediaPlayer"
    (Siehe dazu die Delphi-Hilfe)

(2) Abspielen einer WAV-Datei mittels API-Funktion
    (Siehe Unit "mmSystem")

    sndPlaySound(PChar(FileName),SND_ASYNC);
    sndPlaySound(nil,0);

(3) Tonaufnahme über Mikrophon der Soundkarte mittels API-Funktion
    (Siehe MCI-Befehle der Unit "mmSystem")

    mciSendString(PChar(S),nil,0,handle);

    S=Stringbefehl; nil,0=leerer Datenpuffer; handle=Fensternummer (=0)

(3a) Festlegung des WAV-Formates mittels String, z.B.

    FS := 'BitPerSample 8 Channels 1 SamplePerSec 22050 BytesPerSec 22050';

(3b) Start der Aufzeichnung ("MySound" ist der Aliasname des Audiomediums)

    mciSendString('Open New Type WaveAudio Alias MySound',nil,0,handle);
    mciSendString(PChar('Set MySound Time Format MS ' + FS),nil,0,handle);
    mciSendString('Record MySound',nil,0,handle);

(3c) Ende der Aufzeichnung

    mciSendString('Stop MySound',nil,0,handle);

(3d) Speichern der WAV-Datei

    mciSendString(PChar('Save MySound ' + FileName),nil,0,handle);
    mciSendString('Close MySound',nil,0,handle);
============================================================================== }
```

```
interface

uses Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
     Forms, Dialogs, mmSystem, ExtCtrls, StdCtrls, ShellAPI;

type
  TForm20 = class(TForm)
    Timer1: TTimer;
    Panel1: TPanel;
    Panel2: TPanel;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Button5: TButton;
    Button6: TButton;
    Label1: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    OpenDialog1: TOpenDialog;
    Panel3: TPanel;
    Bevel1: TBevel;
    Bevel2: TBevel;

    procedure FormCreate(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure Panel1Click(Sender: TObject);
    procedure Panel3Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure Button6Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);

  private { Private declarations }
  public  { Public declarations }
end;

var Form20 : TForm20;


implementation
{$R *.dfm}

const S1 : String = 'BITSPERSAMPLE 8 '     +     // 8 Bit
                    'CHANNELS 1 '          +     // MONO
                    'SAMPLESPERSEC 22050 ' +     // 22,050 KHz
                    'BYTESPERSEC 22050 ' ;       // 22050 Bytes/s

const S2 : String = 'BITSPERSAMPLE 16 '    +     // 16 Bit
                    'CHANNELS 2 '          +     // STEREO
                    'SAMPLESPERSEC 22050 ' +     // 22,050 KHz
                    'BYTESPERSEC 88200 ' ;       // 88200 Bytes/s

// BytesPerSec = (BitsPersample div 8) * Channels * SamplesPerSec


var   Verz,SName,FS: String;
      Z: Integer;
      Name0: String = 'sound.wav';
      RecFlag: Boolean = False;
      Vista: Boolean;
```

```
procedure IsSound;
// Soundkarten - Test
var Test: Boolean;
    N: Integer;
begin
  N := WaveOutGetNumDevs;
  Test := (WaveOutGetNumDevs > 0);
  if Test then ShowMessage('Soundkarte vorhanden (' + IntToStr(N) + ')' );
  if Not Test then ShowMessage('Soundkarte NICHT vorhanden !');
end;

function GetSys: String;
// Ermittelt das Betriebssystem
var VersInfo: TOSVersionInfo;
begin
   VersInfo.dwOSVersionInfoSize := SizeOf(VersInfo);
   GetVersionEx(VersInfo);
   Result := ' SystemId. = ' + IntToStr(VersInfo.dwPlatformId) +
             ',  VersionsNr. = ' +
             IntToStr(VersInfo.dwMajorVersion) + '.' +
             IntToStr(VersInfo.dwMinorVersion) + ' ';
end;

function ExecuteFile(const FileName, Params, DefaultDir: string;
                     ShowCmd: Integer): THandle;
// Führt ein Programm oder eine Datei aus
// Wenn Integer(Handle) < 32, dann ist die Ausführung misslungen
var zFileName, zParams, zDir: array[0..79] of Char;
begin
  Result := ShellExecute(Application.MainForm.Handle,
                         nil,
                         StrPCopy(zFileName, FileName),
                         StrPCopy(zParams, Params),
                         StrPCopy(zDir, DefaultDir),
                         ShowCmd);
end;

procedure FitForm(F :TForm);
// Anpassung des Formulares an die Monitorauflösung
const SW: Integer = 1024;
      SH: Integer = 768;
      FS: Integer = 96;
      FL: Integer = 120;
var   X,Y,K: Integer;
      Z: Real;
begin
  with F do begin
    Scaled := True;
    X  := Screen.Width;
    Y  := Screen.Height;
    K  := Font.PixelsPerInch;
    Z  := Y / X;
    if Z <> 0.625 then ScaleBy(X,SW)
                  else ScaleBy(Y,SH);
    if (K <> FS)  then ScaleBy(FS,K);
    Width  := Form20.Bevel2.Width  + 2 * Form20.Bevel2.Left;
    Height := Form20.Bevel2.Height + 10 * Form20.Bevel2.Top;
  end;
end;

procedure TForm20.FormCreate(Sender: TObject);
begin
  FitForm(Form20);
  Form20.Color := RGB(150,160,200);
end;
```

```
procedure TForm20.FormActivate(Sender: TObject);
var L: String;
begin
  L := GetSys;
  Vista := False;
  if Pos('6',L) > 0 then Vista := True;
  FS := S1;
  if FS = S1 then Panel3.Caption := 'Rec.Format: Mono - 8 Bit - 22050 KHz';
  if FS = S2 then Panel3.Caption := 'Rec.Format: Stereo - 16 Bit - 22050 KHz';
  GetDir(0,Verz);
  Edit1.Font.Size := 6;
  Edit1.Text := Trim(LowerCase(Verz));
  Edit2.Text := 'sound';
  SName := Verz + '\' + Name0;
  Z  := 0;
  Panel2.Caption := ' Sec : ' + IntToStr(Z);
  Timer1.Enabled := False;
  RecFlag := False;
end;

procedure TForm20.Timer1Timer(Sender: TObject);
// Timer-Anzeige
begin
   Z := Z + 1;
   Panel2.Caption := ' Sec : ' + IntToStr(Z);
end;

procedure TForm20.Panel1Click(Sender: TObject);
// Sounddatei laden
var V,N,E: String;
begin
  if Timer1.Enabled and RecFlag then begin
     ShowMessage('Zuerst STOP-RECORD-SAVE drücken !'); Exit;
  end;
  if Timer1.Enabled and Not RecFlag then begin
     ShowMessage('Zuerst STOP-PLAY drücken !');
     Exit;
  end;
  with Form20.OpenDialog1 do begin
       InitialDir  := Verz;
       Filter      := 'WAV-Sounddateien |*.wav' ;
       DefaultExt  := '.wav';
       Options     := [ofFileMustExist];
       FileName    := '';
       if Execute then begin
          V := Trim(LowerCase(ExtractFilePath(FileName)));
          N := Trim(LowerCase(ExtractFileName(FileName)));
          E := Trim(LowerCase(ExtractFileExt(FileName)));
          Verz := V;
          Edit2.Text := Copy(N,1,Pos('.',N)-1);
       end;
  end;
  if Verz[Length(Verz)] = '\' then Verz := Copy(Verz,1,Length(Verz)-1);
  Edit1.Text := Trim(LowerCase(Verz));
  Button2.SetFocus;
end;

procedure TForm20.Panel3Click(Sender: TObject);
// Soundkarte testen
begin
  if Timer1.Enabled and RecFlag then begin
     ShowMessage('Zuerst STOP-RECORD-SAVE drücken !'); Exit;
  end;
  if Timer1.Enabled and Not RecFlag then begin
     ShowMessage('Zuerst STOP-PLAY drücken !'); Exit;
  end;
  IsSound;
end;
```

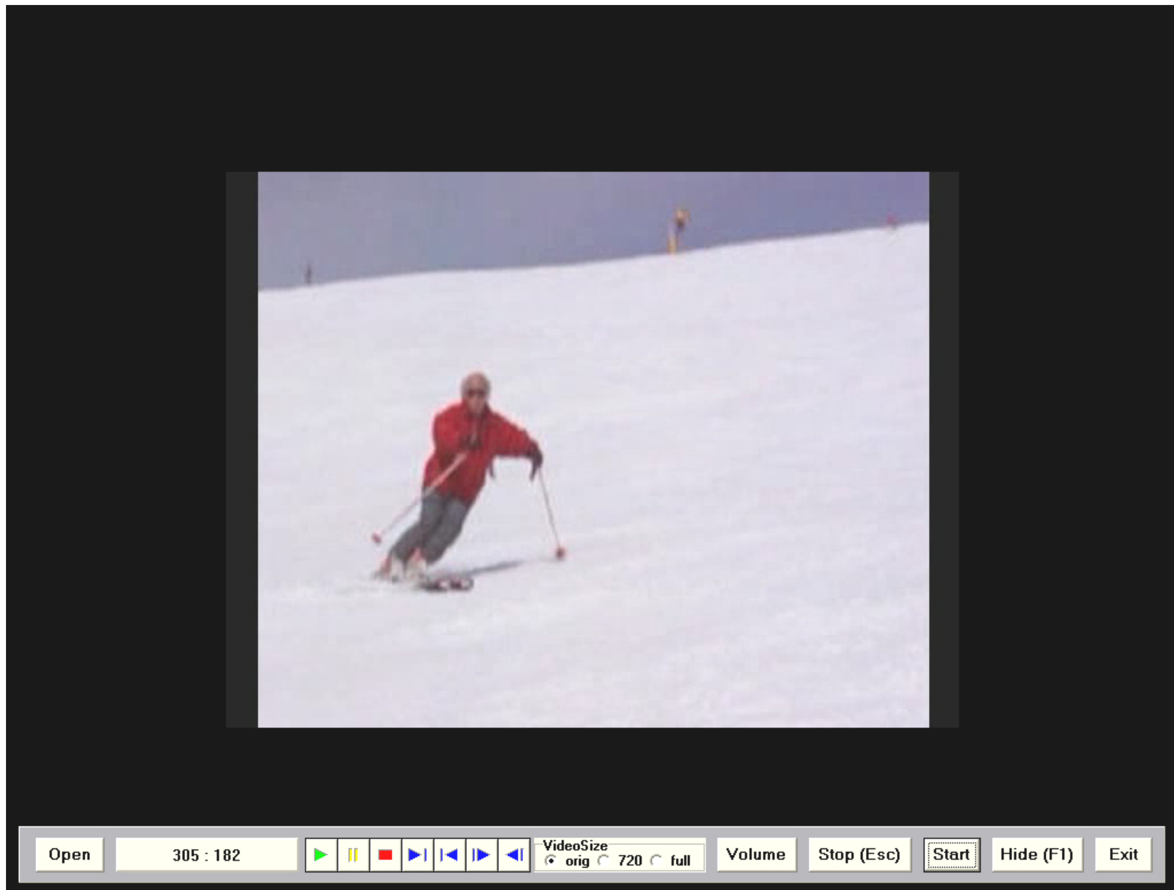
```
procedure TForm20.Button1Click(Sender: TObject);
// Ton - Aufzeichnung - Start
var Q: TPoint;
begin
  if Timer1.Enabled and RecFlag then begin
     ShowMessage('Zuerst STOP-RECORD-SAVE drücken !');
     Exit;
  end;
  if Timer1.Enabled and Not RecFlag then begin
     ShowMessage('Zuerst STOP-PLAY drücken !');
     Exit;
  end;
  if Trim(Edit2.Text) = '' then begin
     ShowMessage('Zuerst einen Dateinamen eingeben !');
     Exit;
  end;

  Button3.SetFocus;
  Q := Mouse.CursorPos;
  Q.X := Form20.Left + Button3.Left + Button3.Width div 2;
  Mouse.CursorPos := Q;

  sndPlaySound(nil,0);

  Z := 0;
  RecFlag := True;
  Timer1.Enabled := True;

  mciSendString('OPEN NEW TYPE WAVEAUDIO ALIAS mysound', nil, 0, 0);
  mciSendString(PChar('SET mysound TIME FORMAT MS ' + FS),nil, 0, 0);
  mciSendString('RECORD mysound', nil, 0, 0);
end;

procedure TForm20.Button2Click(Sender: TObject);
// WAVE-Datei laden und abspielen
var S,T,N: string;
    P: Integer;
    Q: TPoint;
begin
  if Timer1.Enabled and RecFlag then begin
     ShowMessage('Zuerst STOP-RECORD-SAVE drücken !');
     Exit;
  end;
  if Timer1.Enabled and Not RecFlag then begin
     ShowMessage('Zuerst STOP-PLAY drücken !');
     Exit;
  end;
  if Trim(Edit2.Text) = '' then begin
     ShowMessage('Zuerst einen Dateinamen eingeben !');
     Exit;
  end;

  Button6.SetFocus;
  Q := Mouse.CursorPos;
  Q.X := Form20.Left + Button6.Left + Button6.Width div 2;
  Mouse.CursorPos := Q;
  Z := 0;
  Panel2.Caption := ' Sec : ' + IntToStr(Z);
  sndPlaySound(nil,0);
  N := Trim(LowerCase(Edit2.Text));
  T := ExtractFileName(N);
  if T = '' then T := 'sound.wav';
  P := Pos('.',T);
  if P > 0 then T := Copy(T,1,P-1);
  Edit2.Text := T;
  Name0 := T + '.wav';
  SName := Verz + '\' + Name0;
```

```
  if Not FileExists(SName) then begin
     with Form20.OpenDialog1 do begin
       InitialDir  := Verz;
       Filter      := 'WAVE-Sounddateien |*.wav' ;
       DefaultExt  := '';
       Options     := [ofFileMustExist];
       FileName    := '';
       if Execute then begin
          N := Trim(LowerCase(ExtractFileName(FileName)));
          SName := Trim(LowerCase(FileName));
       end
       else Exit;
     end;
  end;

  Name0 := N;
  P := Pos('.',N);
  if P > 0 then N := Copy(N,1,P-1);
  Edit2.Text := N;

  Timer1.Enabled := True;

  S := SName;
  sndPlaySound(PChar(S),SND_ASYNC);
{
  S := 'OPEN "' + SName + '" TYPE WAVEAUDIO ALIAS mysound';
  mciSendString(PChar(S), nil, 0, 0);
  mciSendString('PLAY mysound FROM 0', nil, 0, 0);
  mciSendString('CLOSE mysound', nil, 0, 0);
}
end;

procedure TForm20.Button3Click(Sender: TObject);
// WAVE-Datei Aufnahme-stoppen und speichern
var S,T,N: string;
    P: Integer;
begin
  if Timer1.Enabled and Not RecFlag then begin
     ShowMessage('Zuerst STOP-PLAY drücken !');
     Exit;
  end;
  if Not RecFlag then begin
     ShowMessage('Zuerst START-RECORD drücken !');
     Exit;
  end;
  RecFlag := False;
  Timer1.Enabled := False;
  Z := 0;
  Panel2.Caption := ' Sec : ' + IntToStr(Z);
  sndPlaySound(nil,0);
  mciSendString('STOP mysound', nil, 0, 0);

  N := Trim(LowerCase(Edit2.Text));
  T := ExtractFileName(N);
  if T = '' then T := 'sound.wav';
  P := Pos('.',T);
  if P > 0 then T := Copy(T,1,P-1);
  Edit2.Text := T;
  Name0 := T + '.wav';
  SName := Verz + '\' + Name0;
  S := 'SAVE mysound "' + SName + '"';
  mciSendString(PChar(S), nil, 0, 0);
  mciSendString('CLOSE mysound', nil, 0, 0);
end;
```

```
procedure TForm20.Button4Click(Sender: TObject);
// Lautsprecher-Ansteuerung
var H: THandle;
begin
  if Timer1.Enabled and RecFlag then begin
     ShowMessage('Zuerst STOP-RECORD-SAVE drücken !');
     Exit;
  end;
  if Timer1.Enabled and Not RecFlag then begin
     ShowMessage('Zuerst STOP-PLAY drücken !');
     Exit;
  end;
  if Vista then begin
    H := ExecuteFile('sndvol.exe','','',SW_SHOWNORMAL);
    if Integer(H) < 32 then ShowMessage('Datei "sndvol.exe" nicht gefunden');
  end
  else begin
    H := ExecuteFile('sndvol32.exe','','',SW_SHOWNORMAL);
    if Integer(H) < 32 then ShowMessage('Datei "sndvol32.exe" nicht gefunden');
  end;
end;

procedure TForm20.Button6Click(Sender: TObject);
// Abspielen stoppen
var Q: TPoint;
begin
  if Timer1.Enabled and RecFlag then begin
     ShowMessage('Zuerst STOP-RECORD-SAVE drücken !');
     Exit;
  end;
  if NOT Timer1.Enabled and Not RecFlag then begin
     ShowMessage('Zuerst START-PLAY drücken !');
     Exit;
  end;
  Button2.SetFocus;
  Q := Mouse.CursorPos;
  Q.X := Form20.Left + Button2.Left + Button2.Width div 2;
  Mouse.CursorPos := Q;

  Timer1.Enabled := False;
  Z := 0;
  Panel2.Caption := ' Sec : ' + IntToStr(Z);
  sndPlaySound(nil,0);
end;

procedure TForm20.Button5Click(Sender: TObject);
// Programm beenden
begin
  Timer1.Enabled := False;
  sndPlaySound(nil,0);
  Form20.Close;
end;

procedure TForm20.FormClose(Sender: TObject; var Action: TCloseAction);
// Programm beenden
begin
  Timer1.Enabled := False;
  sndPlaySound(nil,0);
end;

end.
```

[08] Player für Videodateien (*pauvideo*)

Das Programm "***pauvideo***" dient dem Abspielen von beliebigen Video-Dateien in den Formaten mpg, avi, wmv und asf.




```
unit pauvideo_u;
// Videoplayer (c) H. Paukert

interface
uses Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
     Forms, Dialogs, mmSystem, ExtCtrls, StdCtrls, ShellAPI, MPlayer;

type
  TForm20 = class(TForm)
    Timer1: TTimer;
    Panel1: TPanel;
    Panel2: TPanel;
    Panel3: TPanel;
    RadioGroup1: TRadioGroup;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Button5: TButton;
    Button6: TButton;
    OpenDialog1: TOpenDialog;
    MediaPlayer1: TMediaPlayer;
```

```
    procedure FormCreate(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure FormKeyUp(Sender: TObject; var Key: Word;
                        Shift: TShiftState);
    procedure Timer1Timer(Sender: TObject);
    procedure MediaPlayer1Click(Sender: TObject; Button: TMPBtnType;
                                var DoDefault: Boolean);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure Button6Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private { Private declarations }
  public  { Public declarations }
end;

var  Form20 : TForm20;

implementation
{$R *.dfm}

var   Verz, VName: String;
      FW,FH: Integer;
      VideoSize: Integer;
      VideoLen : Int64;
      VideoFra : Int64;
      THide: Boolean;
      BShow: Boolean;
      Vista: Boolean;

function GetSys: String;
// Ermittelt das Betriebssystem
var VersInfo: TOSVersionInfo;
begin
   VersInfo.dwOSVersionInfoSize := SizeOf(VersInfo);
   GetVersionEx(VersInfo);
   Result := ' SystemId. = ' + IntToStr(VersInfo.dwPlatformId) +
             ',  VersionsNr. = ' +
             IntToStr(VersInfo.dwMajorVersion) + '.' +
             IntToStr(VersInfo.dwMinorVersion) + ' ';
end;

function ExecuteFile(const FileName, Params, DefaultDir: string;
                     ShowCmd: Integer): THandle;
// Führt ein Programm oder eine Datei aus
// Wenn Integer(Handle) < 32 dann ist die Ausführung misslungen
var zFileName, zParams, zDir: array[0..79] of Char;
begin
  Result := ShellExecute(Application.MainForm.Handle,
                         nil,
                         StrPCopy(zFileName, FileName),
                         StrPCopy(zParams, Params),
                         StrPCopy(zDir, DefaultDir),
                         ShowCmd);
end;
```

```
procedure HideTaskBar;
// Fullscreen OHNE Taskleiste
begin
  with Form20 do begin
       Align       := alNone;
       BorderStyle := bsNone;
       Left        := 0;
       Top         := 0;
       Width       := Screen.Width;
       Height      := Screen.Height;
       FormStyle   := fsNormal;
       WindowState := wsMaximized;
       Form20.Panel1.Top := Screen.Height - Form20.Panel1.Height - 10;
  end;
  THide := True;
end;


procedure ShowTaskBar;
// Fullscreen MIT Taskleiste
begin
  with Form20 do begin
       BorderStyle := bsSizeable;
       Left        := 0;
       Top         := 0;
       Width       := FW;
       Height      := FH;
       FormStyle   := fsNormal;
       WindowState := wsMaximized;
       Align       := alClient;
       Form20.Panel1.Top  := Form20.ClientHeight - Form20.Panel1.Height - 10;
  end;
  THide := False;
end;


procedure HideButtons;
// Visuelle Objekte unsichtbar
begin
  Form20.Panel1.Visible := False;
  Form20.Panel1.SendToBack;
  Application.ProcessMessages;
end;


procedure ShowButtons;
// Visuelle Objekte sichtbar
begin
  Form20.Panel1.Visible := True;
  Form20.Panel1.BringToFront;
  Application.ProcessMessages;
end;
```

```
procedure FitForm(F :TForm);
// Anpassung des Formulares an die Monitorauflösung
const SW: Integer = 1024;
      SH: Integer = 768;
      FS: Integer = 96;
      FL: Integer = 120;
var   X,Y,K: Integer;
      Z: Real;
begin
  with F do begin
    Scaled := True;
    X  := Screen.Width;
    Y  := Screen.Height;
    K  := Font.PixelsPerInch;
    Z  := Y / X;
    if Z <> 0.625 then ScaleBy(X,SW) else ScaleBy(Y,SH);
    if (K <> FS)  then ScaleBy(FS,K);
    WindowState := wsMaximized;
  end;
end;

procedure TForm20.FormCreate(Sender: TObject);
// Initialisierungen
begin
  FitForm(Form20);
  Form20.Color := clBlack;
  Form20.Panel3.Color := clBlack;
  FW := Form20.Width;
  FH := Form20.Height;
end;

procedure TForm20.FormActivate(Sender: TObject);
// Initialisierungen
var L: String;
begin
  L := GetSys;
  Vista := False;
  if Pos('6',L) > 0 then Vista := True;
  GetDir(0,Verz);
  Panel2.Caption := ' 0 : 0 ';
  Timer1.Enabled := False;
  VName := '';
  Panel1.Left := (Form20.ClientWidth - Panel1.Width) div 2;
  Panel1.Top  := Form20.ClientHeight - Panel1.Height - 10;
  THide := False;
  BShow := True;
  ShowButtons;
  Button1.SetFocus;
end;

procedure TForm20.FormKeyUp(Sender: TObject; var Key: Word;
                            Shift: TShiftState);
// Schalter sichtbar machen
begin
   if Key = vk_F1 then BShow := NOT BShow;
   if BShow then ShowButtons else HideButtons;
end;
```

```
procedure MPlayerOff;
// Mediaplayer ausschalten
begin
  try
    if NOT Form20.MediaPlayer1.Enabled then Exit;
    if (Form20.MediaPlayer1.Mode <> mpPlaying) and
       (Form20.MediaPlayer1.Mode <> mpStopped) then Exit;
    Form20.Mediaplayer1.AutoEnable := True;
    Form20.MediaPlayer1.Stop;
    Form20.MediaPlayer1.Close;
    Form20.MediaPlayer1.Enabled := False;
    Form20.Timer1.Enabled := False;
    Form20.Panel2.Caption := '0 : 0';
    Form20.Panel3.Visible := False;
  except
  end;
end;

procedure MPlayerON(S: String);
// Mediaplayer einschalten und Videodatei S abspielen
var x,y: Integer;
    Rect0 : TRect;
    w0,h0,w1,h1: Integer;
    fk: Real;
begin
  if (Pos('.avi',S) > 0) or (Pos('.mpg',S) > 0) or
     (Pos('.wmv',S) > 0) or (Pos('.asf',S) > 0) then begin
      VideoSize := Form20.RadioGroup1.ItemIndex + 1;
      x := 0;
      y := 0;
      Form20.MediaPlayer1.Enabled := True;
      Form20.MediaPlayer1.AutoEnable := False;
      Form20.MediaPlayer1.FileName := S;
      Form20.MediaPlayer1.Open;
      Form20.MediaPlayer1.Wait := False;
      Form20.MediaPlayer1.Notify := False;
      Form20.MediaPlayer1.Display := Nil;
      Rect0 := Form20.MediaPlayer1.DisplayRect;
      w0 := (Rect0.Right - Rect0.Left);
      h0 := (Rect0.Bottom - Rect0.Top);

      if VideoSize = 1 then begin
         w1 := w0;
         h1 := h0;
      end;
      if VideoSize = 2 then begin
         w1 := 720;
         fk := w1 / w0;
         h1 := Round(h0*fk);
      end;
      if VideoSize = 3 then begin
         h1 := Screen.Height;
         fk := h1 / h0;
         w1 := Round(w0*fk);
      end;

      Form20.Panel3.Width  := w1;
      Form20.Panel3.Height := h1;
```

```
      if (w1 < Form20.Width) and (VideoSize < 3) then begin
         x := (Form20.Width - w1) div 2;
      end;
      if (h1 < Form20.Height) and (VideoSize < 3) then begin
         y := (Form20.Height - h1) div 2;
      end;
      if (VideoSize = 3) then begin
         if (w1 < Screen.Width) then begin
            x := (Screen.Width - w1) div 2;
         end;
      end;

      Form20.Panel3.Left := x;
      Form20.Panel3.Top  := y;
      Form20.Panel3.Visible := True;
      Form20.Panel3.BringToFront;
      Form20.MediaPlayer1.Display := Form20.Panel3;
      Form20.MediaPlayer1.DisplayRect := Rect(0,0,w1,h1);
      VideoLen := Form20.MediaPlayer1.Length;
      VideoFra := VideoLen div 10;
      Form20.MediaPlayer1.Frames := VideoFra;
      Form20.Timer1.Interval := 1000;
      Form20.Timer1.Enabled := True;
      Form20.MediaPlayer1.Play;
  end;
end;

procedure TForm20.Timer1Timer(Sender: TObject);
// MediaPlayer-Position anzeigen
begin
   if (GetAsyncKeyState(VK_ESCAPE) <> 0) then begin
      MPlayerOff;
      BShow := True;
      ShowButtons;
      Exit;
   end;
   Form20.Panel2.Caption := IntToStr(Form20.MediaPlayer1.Position) + ' : ' +
                            IntToStr(VideoLen);
end;

procedure TForm20.MediaPlayer1Click(Sender: TObject; Button: TMPBtnType;
                                    var DoDefault: Boolean);
// Aktivierung der Playertasten
begin
  with MediaPlayer1 do begin
    Timer1.Enabled := False;
    Stop;
    case Button of
      btPlay : Timer1.Enabled := True;
      btNext : Position := Length - 1;
      btPrev : Position := 0;
      btStep : Position := Position + VideoFra;
      btBack : Position := Position - VideoFra;
    end;
    Form20.Panel2.Caption := IntToStr(Position) + ' : ' + IntToStr(Length);
    Application.ProcessMessages;
  end;
end;
```

```
procedure TForm20.Button1Click(Sender: TObject);
// Videofile öffnen und abspielen
var V,N,E: String;
begin
  MPlayerOff;
  with Form20.OpenDialog1 do begin
       InitialDir := Verz;
       Filter := 'WMV-Videos |*.wmv|' +
                 'MPG-Videos |*.mpg|';
       DefaultExt  := '.wmv';
       Options     := [ofFileMustExist];
       FileName    := '';
       if Execute then begin
          VName := FileName;
          V := Trim(LowerCase(ExtractFilePath(FileName)));
          N := Trim(LowerCase(ExtractFileName(FileName)));
          E := Trim(LowerCase(ExtractFileExt(FileName)));
          Verz := V;
       end;
  end;
  if Verz[Length(Verz)] = '\' then Verz := Copy(Verz,1,Length(Verz)-1);
  if NOT THide then HideTaskbar;
  MPlayerON(VName);
  BShow := True;
  ShowButtons;
end;


procedure TForm20.Button2Click(Sender: TObject);
// Lautsprecher-Ansteuerung
var H: THandle;
begin
  if Vista then begin
    H := ExecuteFile('sndvol.exe','','',SW_SHOWNORMAL);
    if Integer(H) < 32 then ShowMessage('Datei "sndvol.exe" nicht gefunden');
  end
  else begin
    H := ExecuteFile('sndvol32.exe','','',SW_SHOWNORMAL);
    if Integer(H) < 32 then ShowMessage('Datei "sndvol32.exe" nicht gefunden');
  end;
end;


procedure TForm20.Button3Click(Sender: TObject);
// Geöffnetes Video stoppen
begin
  if VName = '' then begin
     ShowMessage(' Bitte zuerst eine Videodatei öffnen ! ');
     Exit;
  end;
  MPlayerOff;
end;
```

```
procedure TForm20.Button4Click(Sender: TObject);
// Geöffnetes Video starten
begin
  if VName = '' then begin
     ShowMessage(' Bitte zuerst eine Videodatei öffnen ! ');
     Exit;
  end;
  MPlayerOn(VName);
  BShow := True;
  ShowButtons;
end;


procedure TForm20.Button5Click(Sender: TObject);
// Schalter unsichtbar
begin
  BShow := False;
  HideButtons;
end;


procedure TForm20.Button6Click(Sender: TObject);
// Programm beenden
begin
  Timer1.Enabled := False;
  ShowTaskbar;
  MPlayerOff;
  Form20.Close;
end;


procedure TForm20.FormClose(Sender: TObject; var Action: TCloseAction);
// Programm beenden
begin
  Timer1.Enabled := False;
  ShowTaskbar;
  MPlayerOff;
end;


end.
```