

M T O O L S

Drei Multimedia-Werkzeuge

© Herbert Paukert

(Letztes Update im Jänner 2018)

- [A] WINDOWSPRO. Verwaltung von Windows-Tasks. (- 02 -)**
- [B] IMAGEPRO. Einfache Grafikverarbeitung. (- 11 -)**
- [C] MEDIAPRO. Multimediale Präsentationen. (- 17 -)**

Dieses Skriptum ist der letzte Teil eines Programmierkurses in DELPHI. Es werden von jedem Programm eine Bedienungsanleitung und eine kurze Auflistung von ausgewählten Routinen angegeben.

[A] WINDOWSPRO. Verwaltung von Windows-Tasks.



Beschreibung des Programms

Das Programm dient der einfachen Verwaltung von WINDOWS-Tasks.
Es besteht aus einem Textfeld, einem Inputfeld und sechs Schaltern:

- < ? > Hilfstext ein- und ausblenden
- < *ListProc* > erzeugt im Textfeld eine Liste von den Prozessen (Tasks),
die im Betriebssystem aktuell ablaufen.
- [*Input . . .*] ermöglicht die Auswahl einer beliebigen Datei (Client).
- < *GetProc* > ermittelt das zu einer Client-Datei registrierte Server-Programm.
- < *WMediaPlayer* > öffnet den Windows-Mediaplayer.
- < *KillProc* > beendet den jeweils aufgerufenen Task.
- < *Exit* > schließt das Programm.

Mit einem Doppelklick in das Inputfeld wird eine Datei-Auswahlbox geöffnet, und es kann der Name einer beliebigen Datei in das Inputfeld transferiert werden. Wenn es sich dabei um eine Datendatei (Clients: Text, Grafik, Video, usw.) handelt, dann wird mit dem Schalter <*GetProc*> jenes Programm (Server) ermittelt und ausgeführt, welches in WINDOWS zum Öffnen der Datendatei registriert ist.

Zusätzlich werden beim Programmstart Systemdaten des Computers ausgelesen. (CPU-Nummer, Betriebssystem-Version, usw.)

Alle Routinen des Programms

Das Programm *WINDOWSPRO* enthält die Hauptunit *windowspro_u* und die Nebenunit *hptools_u*, welche u.a. Routinen zur Taskverwaltung enthält. Es folgt das komplette Listing des Programms.

unit hptools_u;

```
// Windows-Werkzeug-Routinen (c) Herbert Paukert

interface
uses SysUtils, Windows, Classes, Forms, ShellAPI, TLHelp32;

var WinTempVerz: String = '';   // Temporäres Windows-Verzeichnis
    StrList: TStringList;       // Stringliste der Windows-Prozesse
    SLmax : Integer = 0;       // Zähler für die Stringliste

function get_unique_string: string;
function GetTempPathEx: String;
function GetSys: String;
function GetXP: Boolean;

function ExecuteFile(const FileName, Params, DefaultDir: string;
                      ShowCmd: Integer): THandle;
function IsExeRunning(const ExeFileName: string): boolean;
function KillExe(const ExeFileName: string): integer;
function GetExeForFile(const FileName: String): String;
function KillTask(ExeFileName: string): integer;
function KillServProg(FileName: String): Integer;

procedure ListProc;

implementation
```

function read_cpu_string: string;

```

type ba = array[0..3] of char;
var s1,s2,s3: Cardinal;
    Model: Cardinal;
begin
  Result := '';
  asm
    pushad
    xor   eax,eax
    db   $0f,$a2 // cpuid
    mov  s1,ebx
    mov  s2,edx
    mov  s3,ecx
    mov  eax,1
    db   $0f,$a2 // cpuid
    mov  Model,eax
  popad
end;
Result := string(ba(s1)) + string(ba(s2)) + string(ba(s3));
if Result <> 'GenuineIntel' then Result:='Other';
Result := Result + IntToHex(Model,8);
end;

```

function read_hw_string: string;

```

var HwInfo: tagHW_PROFILE_INFOA;
begin
  GetCurrentHwProfile(HwInfo);
  Result := IntToStr(HwInfo.dwDockInfo) + ':' +
           string(HwInfo.szHwProfileName) + ':' +
           string(HwInfo.szHwProfileGuid);
end;

```

function get_unique_string: string;

```

var S: string;
begin
  S := read_cpu_string + ':' + read_hw_string;
  Result := Trim(S);
end;

```

function GetTempPathEx: String;

```

// Ermittlung des lokalen, temporären Windows-Ordners
var len,templen: Cardinal;
    temp: array of Char;
    s: String;
begin
  WinTempVerz := '';
  result := '';
  len := 20;
  SetLength(temp,len);
  templen := GetTempPath(len,PChar(temp));
  if templen > len then begin
    SetLength(temp,templen+1);
    GetTempPath(templen+1,PChar(temp));
    s := trim(String(temp));
    if s[Length(s)] = '\' then s := Copy(s,1,Length(s)-1);
    result := s;
    WinTempVerz := s;
  end;
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 GetXP;

```
// Prüft WINDOWS XP
var L: String;
begin
  L := GetSys;
  if Pos('5',L) > 0 then Result := True
    else Result := False;
end;
```

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

```
// Führt ein Programm oder eine Datei "FileName" aus
// mit optionalem Parameter "Params"
// mit optionalem Zugriffs-Verzeichnis
// mit ShowCmd = SW_SHOWNORMAL, SW_SHOWMAXIMIZED, SW_SHOWMINIMIZED
// 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;
```

function IsExeRunning(const ExeFileName: string): boolean;

```
// Einen laufenden Prozess identifizieren
var
  h: THandle;
  p: TProcessEntry32;
begin
  Result := False;
  p.dwSize := SizeOf(p);
  h := CreateToolHelp32Snapshot(TH32CS_SnapProcess, 0);
  try
    Process32First(h,p);
    repeat
      Result := AnsiUpperCase(ExeFileName) = AnsiUpperCase(p.szExeFile);
    until Result or (not Process32Next(h, p));
  finally
    CloseHandle(h);
  end;
end;
```

function KillExe(const ExeFileName: string): integer;

```
// Einen laufenden Prozess (Task) entfernen
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(ExtractFileName(FProcessEntry32.szExeFile)) =
      UpperCase(ExeFileName))
    or (UpperCase(FProcessEntry32.szExeFile) =
      UpperCase(ExeFileName)))
    then Result := Integer(TerminateProcess(OpenProcess(PROCESS_TERMINATE,
      BOOL(0), FProcessEntry32.th32ProcessID), 0));
    ContinueLoop := Process32Next(FSnapshotHandle,FProcessEntry32);
  end;
  CloseHandle(FSnapshotHandle);
end;
```

```
function GetExeForFile(const FileName: String): String;
// Zu einer Datendatei (Client) das registrierte Windows-Programm (Server)
erkennen
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 KillTask(ExeFileName: string): integer;
// EXE-Programm beenden
var PName: String;
begin
  PName := Trim(ExtractFileName(ExeFileName));
  if PName <> '' then begin
    // ShowMessage('Kill: ' + PName);
    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 ListProc;
// Liste der laufenden Prozesse (Taskmanager)
var hSnap: THandle;
    ProcEntry: TProcessEntry32;
    s,t: String;
begin
  StrList.Clear;
  hSnap := CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
  if (hSnap <> INVALID_HANDLE_VALUE) then begin
    ProcEntry.dwSize := SizeOf(ProcessEntry32);
    if (Process32First(hSnap, ProcEntry)) then begin
      s := ProcEntry.szExeFile;
      t := ExtractFileName(s);
      StrList.Add(t);
      while Process32Next(hSnap, ProcEntry) do begin
        s := ProcEntry.szExeFile;
        t := ExtractFileName(s);
        StrList.Add(t);
      end;
    end;
  end;
  CloseHandle(hSnap);
  SLmax := StrList.Count;
end;
```

```
initialization
  StrList := TStringList.Create;
```

```
finalization
  StrList.Free;
```

```
end.
```

unit windowspro_u;

```

// Prozesse auflisten, ausführen und beenden (c) Herbert Paukert
// Version 20.1

interface
uses Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
    Forms, Dialogs, StdCtrls, Printers, ExtCtrls, ComCtrls, hptools_u;

type
  TForm1 = class(TForm)
    OpenFileDialog: TOpenDialog;
    ListBox1: TListBox;
    Edit1: TEdit;
    Label1: TLabel;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Button5: TButton;
    Button6: TButton;
    Mem1: TMemo;

    procedure FormCreate(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure ListBox1Click(Sender: TObject);
    procedure Edit1Db1Click(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);

  public { Public declarations }
  end;

var Form1: TForm1;

implementation
{$R *.dfm}

var Verz, Ext, FName: String;
    xp: Integer = 10;
    yp: Integer = 10;
    openFlag: Boolean = False;

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;
    V0,V : Real;
begin
  with F do begin
    Scaled := True;
    X := Screen.Width;
    Y := Screen.Height;
    K := Font.PixelsPerInch;
    V0 := SH / SW;
    V := Y / X;
    if V < V0 then ScaleBy(Y,SH)
      else ScaleBy(X,SW);
    if (K <> FS) then ScaleBy(FS,K);
    Position := poDesigned;
    // WindowState := wsMaximized;
  end;
end;

```

procedure TForm1.FormCreate(Sender: TObject);

```
// Grundeinstellungen
var s: String;
begin
  ForceCurrentDirectory := True;
  GetDir(0,Verz);
  FitForm(Form1);
  Form1.Color := RGB(160,180,200);
  s := '(*) Click <ListProc> ' + #13 +
    '(*) DoubleClick <Input>' + #13 +
    '(*) Choose FileName ' + #13 +
    '(*) Click <GetProc> ' + #13 +
    '(*) Click <KillProc> ' + #13 +
    '(*) Click in Listbox ' + #13 +
    ' ';
  Labell1.Caption := s;
end;
```

procedure TForm1.FormActivate(Sender: TObject);

```
// Windows-Parameter auslesen
var x,y,f: Integer;
var h,a,b,s,t,v,z: String;
begin
  Form1.Left := 0;
  Form1.Top := 0;
  Form1.Height := Form1.Top + Form1.Button3.Top + 3* Form1.Button3.Height;
  Form1.Width := 8 * Form1.ListBox1.Left + Form1.ListBox1.Width - 20;
  x := Screen.Width;
  y := Screen.Height;
  if (x = 1536) and (y = 864) then begin
    x := 1920;
    y := 1080;
  end;
  f := Font.PixelsPerInch;
  a := IntToStr(x) + ' x ' + IntToStr(y) + ', FontSize = ' + IntToStr(f);
  b := 'DecimalSeparator = ' + DecimalSeparator;
  t := Trim(GetTempPathEx);
  s := Trim(GetSys);
  h := get_unique_string;
  if GetXP then v := 'Windows XP'
    else v := 'Windows höher als XP';
  z := 'Prozessor-Information:' + #13 + h + #13#13 +
    'Betriebssystem-Information:' + #13 + s + #13 + v + #13#13 +
    'Bildschirm-Auflösung:' + #13 + a + #13 + b + #13#13 +
    'Temporäres Windows-Verzeichnis:' + #13 + t;
  ShowMessage(z);
end;
```

procedure TForm1.ListBox1Click(Sender: TObject);

```
// Einen laufenden Prozess auswählen
// Diese Prozedur ist sicherheitshalber ausgeschaltet
var I: Integer;
    S: String;
begin
  Exit;
  I := ListBox1.ItemIndex;
  S := ListBox1.Items[I];
  Edit1.Clear;
  Edit1.Text := S;
end;
```


procedure TForm1.Edit1DbClick(Sender: TObject);

```
// Einen Dateinamen finden
var T: String;
begin
  Edit1.Clear;
  With OpenFileDialog do begin
    if NOT openFlag then begin
      T := ' Klicken Sie auf den gewünschten Ordner' + #13 +
        ' und dann auf die gewünschte Datei.      ';
      ShowMessagePos(T,xp,yp);
      if Length(Verz) > 3 then FileName := Verz
        else InitialDir := Verz;
    end
  else begin
    InitialDir := Verz;
    FileName := '*.*';
  end;
  openFlag := True;
  Filter := 'Alle Dateien (*.*)|*.*';
  Options := [ofFileMustExist];
  if Execute then begin
    Verz := Trim(ExtractFilePath(FileName));
    Ext := Trim(LowerCase(ExtractFileExt(FileName)));
    FName := Trim(ExtractFileName(FileName));
    Edit1.Text := LowerCase(FName);
  end;
end;
end;
```

procedure ShowListbox;

```
// Alle laufenden Windows-Prozesse anzeigen
var i: Integer;
    s: String;
begin
  ListProc;
  Form1.ListBox1.Clear;
  For i := 0 to SLmax-1 do begin
    s := StrList[i];
    Form1.ListBox1.Items.Add(s);
  end;
end;
```

procedure TForm1.Button1Click(Sender: TObject);

```
// Alle laufenden Windows-Prozesse anzeigen
begin
  Mem1.Visible := False;
  ShowListbox;
end;
```

procedure TForm1.Button2Click(Sender: TObject);

```
// Programm oder Prozess beenden
var s: String;
    i: Integer;
begin
  Mem1.Visible := False;
  s := Trim(Edit1.Text);
  if s = '' then Exit;
  Ext := ExtractFileExt(s);
  if Ext = '.exe' then KillTask(s)
    else KillServProg(s);

  ListBox1.Clear;
  Edit1.Clear;
  Edit1.Text := 'Input...';
  Application.ProcessMessages;
  OpenFlag := False;
end;
```

procedure TForm1.Button3Click(Sender: TObject);

```
// Programm beenden
begin
  Memol.Visible := False;
  Halt;
end;
```

procedure TForm1.Button4Click(Sender: TObject);

```
// Prozess oder Programm identifizieren
var R,S,T,F: String;
    N,Code: Integer;
    H: THandle;
begin
  Memol.Visible := False;
  S := Trim(Edit1.Text);
  R := LowerCase(S);
  if Pos('.exe',R) > 0 then begin
    ShowMessage('Das ist ein "EXE"-File !');
    Exit;
  end;
  T := GetExeForFile(S);
  Edit1.Text := T;
  H := ExecuteFile(S, '', '', SW_SHOWNORMAL);
  if H < 32 then ShowMessage('file not found !');
end;
```

procedure TForm1.Button5Click(Sender: TObject);

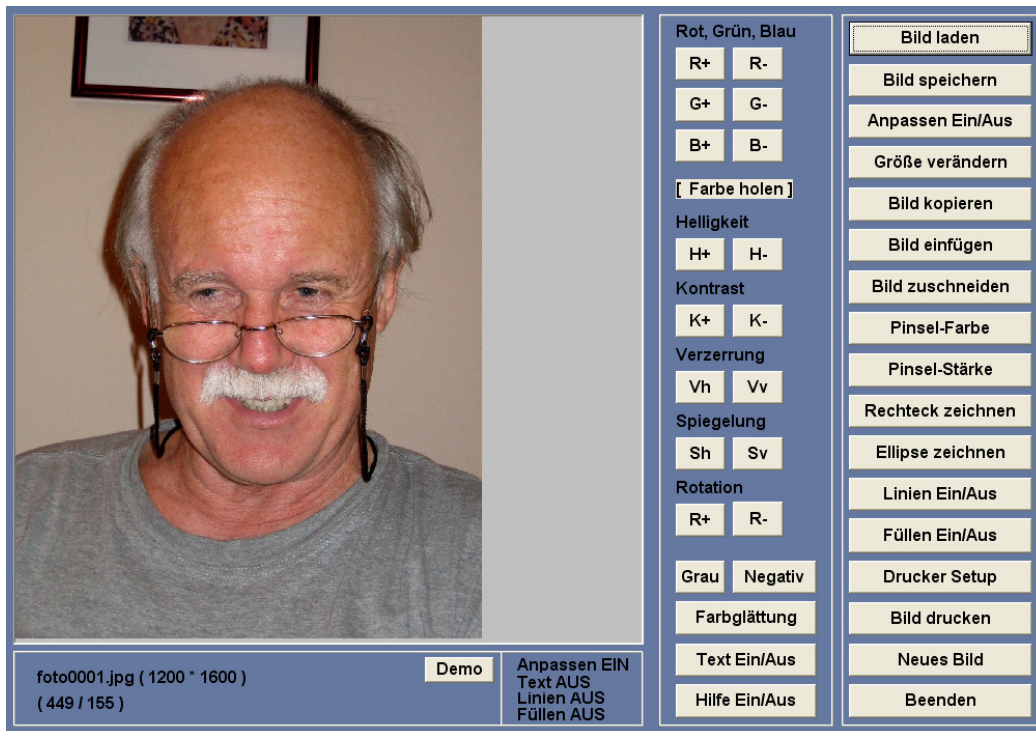
```
// WMPlayer aufrufen
var H: THandle;
    s: String;
    i: Integer;
begin
  Memol.Visible := False;
  if OpenFlag then begin
    ShowMessage('Zuerst laufeneden Prozess mit "KillProc" beenden!');
    Exit;
  end;
  Edit1.Text := '';
  H := ExecuteFile('wmplayer.exe', '', '', SW_SHOWNORMAL);
  if H < 32 then begin
    ShowMessage('"wmplayer.exe" not found !');
    Exit;
  end;
  Edit1.Text := 'wmplayer.exe';
  ShowListbox;
  OpenFlag := True;
end;
```

procedure TForm1.Button6Click(Sender: TObject);

```
// Kurze Hilfe ein/aus
begin
  Memol.Visible := NOT Memol.Visible;
end;
```

end.

[B] IMAGEPRO. Einfache Grafikverarbeitung.



Beschreibung des Programms

- (I) Es können BMP- und JPG-Grafikdateien geladen, modifiziert und gegenseitig konvertiert werden.
- (II) Einfache Mal- und Zeichenfunktionen sind möglich.
- (III) Die Grafiken können positioniert und skaliert ausgedruckt werden (<Setup> und <Drucken>).

Hinweis: Mit Schalter <Demo> wird der Zielordner für Bildeingaben eingestellt: entweder der aktuelle Ordner oder ein interner Demo-Ordner mit Fotos zum Testen der verschiedenen Programm-Funktionen.

Am Anfang muss entweder <Bild laden> oder <Neues Bild> betätigt werden. Dann stehen folgende Funktionen zur Verfügung: Verändern der Farbwerte, Umwandeln in Graustufen, negative Farben, Verändern von Helligkeit und Kontrast, Farbglätten, Spiegeln, Verzerrern, Rotieren, Verkleinern und Vergrößern, Zuschneiden, usw.

 Bereichsmarkierung durch Ziehen der Maus mit gedrückter rechter Maustaste, von links oben nach rechts unten.
 Ein Mausklick löscht immer eine gezogene Markierung.

Malen und Zeichnen erfolgt mit Mausbewegungen bei gedrückter linker Taste. Dazu muss vorher unbedingt der Schalter <Anpassen AUS> gesetzt werden.

Die meisten Befehlsschalter sind nur dann wirksam, wenn das Bild auf wahre Größe eingestellt ist (Anpassen AUS). Die aktuelle Bildposition wird für viele Befehle ganz einfach mit einem linken Mausklick bestimmt.

BEISPIELE:

- [] Markierten Bereich mittels <Bild kopieren> kopieren. Die Hintergrund-Transparenz ist einstellbar. Funktioniert nur, wenn erstens der Schalter auf <Anpassen AUS> gesetzt ist, und zweitens ein Rahmen von links oben nach recht unten markiert ist.
- [] Kopierten Bereich an aktueller Position einfügen. Zuerst <Bild einfügen> betätigen, dann ein Mausklick an der Position. Mit Taste <F1> kann die letzte Bild- oder Text-Einfügung wieder rückgängig gemacht werden.
- [] Der Schalter <Füllen EIN> ermöglicht die Füllung der Grafik ab der doppelt angeklickten Position mit der gewählten Füllfarbe bis zum geschlossenen Rand in der aktuellen Zeichenfarbe. Zuerst Schalter betätigen, dann doppelter Mausklick an der Position. Achtung, bei nicht geschlossenen Bereichsrand rinnt die Farbe aus. Zuletzt Schalter wieder ausschalten!
- [] Der Schalter <Farbglättung> ermöglicht in einem markierten Bereich alle Farbübergänge zu glätten (Weichzeichnen der Konturen). Nach der Markierung kann der Schalter beliebig oft gedrückt werden.

Das Programm hat keine eigene UNDO-Funktion. Es kann aber jederzeit ein bearbeitetes Bild mittels <Strg C> in die Zwischen-Ablage kopiert und von dort wieder mit <Strg V> eingefügt werden.

Mit der Schaltfläche <Farbe holen> kann die Farbe an der vorher kurz angeklickten Bildposition als Zeichenfarbe übernommen werden.

Der Wechselschalter <Text Ein/Aus> ermöglicht Textausgaben an der nachfolgend angeklickten Bildposition. Mit <F1> wird die letzte Textausgabe rückgängig gemacht.

Der Wechselschalter <Linien Ein/Aus> ermöglicht das Zeichnen eines Linienzuges, beginnend mit einem linken Mausklick und dann fortlaufend mit rechten Mausklicks. Dabei wird das Markieren von Bereichen unterbunden. Bereiche können nur dann wieder markiert werden, wenn <Linien Ein/Aus> ausgeschaltet ist.

Mit jedem Mausklick auf einen Bildpunkt werden dessen Koordinaten laufend in einen String gespeichert, der mit <F9> in die Zwischenablage kopiert werden kann. Mit <Strg><F9> hingegen wird dieser String gelöscht. Gespeicherten Koordinaten werden in jeden Texteditor aus der Windows-Zwischenablage mit <Strg V> eingefügt.

Große Grafiken werden mit Schalter <Anpassen Ein/Aus> entweder an das Bildfenster optimal angepasst oder in ihrer wahren Größe angezeigt. Im zweiten Fall wird mittels Bildlaufleisten die Grafik gescrollt.

Beispiel 1: "Passbilder erzeugen"

Die Größe und Position eines Grafikausdruckes kann auf Millimeter genau auf einem A4-Blatt bestimmt werden (z.B. für Passbilder). Dazu sollte mit dem Schalter <Drucker Setup> die höchste Druckerauflösung gesetzt werden. Der zu druckende Bereich muss nun so markiert werden, dass sein Seitenverhältnis $x : y$ ungefähr $1 : 1.20$ beträgt. Dann wird die Grafik mit <Bild zuschneiden> auf diesen Bereich zugeschnitten. Mit Schalter <Drucken> ist die Druckskalierung (ohne wirklich zu drucken) so lange zu wiederholen, bis das Druckbild ungefähr die Maße 3.5×4.2 cm hat. Jetzt kann das Bild endgültig ausgedruckt werden. Der Ausdruck kann am selben Blatt an einer neuen passenden Position wiederholt werden (Papierersparnis).

Beispiel 2: "Bilder clonen"

Es kann um jeden gewünschte Bereich eines Bildes mit der Maus eine geschlossene schwarze Randlinie händisch eingezeichnet werden (Scherer). Nun wird mit Schalter <Füllen EIN> die Füllfarbe Schwarz gewählt. Sodann wird mit linkem Doppelklick auf einen Punkt außerhalb des Bereichs das Äußere des Bereichs schwarz gefärbt. Als nächstes wird ein rechteckiger Rahmen um den Bereich mit der linken und der rechten Maustaste markiert und dann dieser markierte Bereich in den Zwischenspeicher kopiert. Dabei muss Hintergrund-Transparenz eingestellt werden. Wenn dies der Fall ist, dann kann mit dem Schalter <Einfügen> an jeder Position eines neuen oder geladenen Bildes der zwischengespeicherte Bereich so eingefügt werden, dass dort, wo schwarze Farbe ist, der Hintergrund des neuen Bildes erhalten bleibt.

ZUSÄTZE:

Mit <F1> kann die letzte Bild- oder Text-Einfügung wieder rückgängig gemacht werden.

Mit <F2> im geöffneten Bild können Zoom-Effekte getestet werden. In diesem totalen Zoom werden die maximale Zoomstärke (1% bis 1000%) und die Zoomgeschwindigkeit (Zoomschritte) als Parameter ein gegeben. Bei negativer Zoomstärke wird der Zoom wieder rückgängig gemacht.

Mit <F3> kann im geöffneten Bild ein begrenzter Ausschnitt, der vorher markiert worden ist, kleiner oder größer gezoomt werden. Die Zoomstärke wird dabei im Bereich von 1% bis 1000% gewählt. Auch die Zoomgeschwindigkeit (Zoomschritte) muss eingegeben werden. Bei negativer Zoomstärke wird der Zoom wieder rückgängig gemacht.

Mit <F4> kann im geöffneten Bild eine begrenzte Bild Deformation erzeugt werden - und zwar um jenen Punkt, der NACHHER einfach angeklickt wird. Die Deformationsstärke wird dabei zwischen 1 und 100, die Ausdehnung der Deformation wird zwischen 1 und 1000 gewählt. Mit <Strg><F4> wird das Originalbild wiederhergestellt.

Mit <F5> kann im geöffneten Bild in einen markierten Ausschnitt ein weiteres Bild direkt eingefügt werden. (PIP = Picture In Picture). Dabei können durch Parameter Rahmendicke, Rahmenfarbe und Hintergrundtransparenz der eingefügten Bilder gesteuert werden. Mit <Strg><F5> wird das Originalbild wiederhergestellt.

Mit <F6> im geöffneten Bild kann eine einfache Bildzerstörung (Blockmosaik) getestet werden. Mit <Strg><F6> wird das Originalbild wiederhergestellt.

Mit <F7> im geöffneten Bild können die nächsten geöffneten Bilder überblendet werden. Es gibt Schwarzblenden, Kreuzblenden und auch Gleitblenden.

Mit <F8> im geöffneten Bild können alle Bilder des aktuellen Verzeichnisses auf eine konstante Höhe automatisch zugeschnitten und umbenannt werden. Wenn dabei die Bildhöhe 0 eingegeben wird, dann erfolgt nur eine Umbenennung der Bilddateien. Bei der Umbenennung werden die Bilder automatisch ab einer gewählten Startnummer fortlaufend nummeriert. Anstelle der Höhe der Bilder kann auch die Breite der Bilder konstant geändert werden. Das Verhältnis von Bildhöhe zu Bildbreite (aspect ratio) bleibt dabei immer unverändert. Die Routine ist für maximal 1000 Bilddateien vorgesehen.

Mit <F9> wird der Textpuffer von allen angeklickten Bildkoordinaten in die WINDOWS-Zwischenablage kopiert. Mit <Strg><F9> wird der Textpuffer von allen vorher angeklickten Bildkoordinaten gelöscht.

Einige Routinen des Programms

Das Programm *IMAGEPRO* enthält die Hauptunit *paupix_u* und die Nebenunit *gtools_u*. In *gtools_u* sind nützliche Routinen zur Grafikverarbeitung gespeichert. Als Beispiele werden nachfolgende Routinen beschrieben:

procedure PaintImage(I: TImage; F: TColor; X1,Y1,X2,Y2 : Integer);

```
// Füllt ein Image-Rechteck mit einer Farbe
begin
  with I.Canvas do begin
    Pen.Color := F;
    Brush.Color := F;
    Brush.Style := bsSolid;
    FillRect(Rect(X1,Y1,X2,Y2));
    Brush.Style := bsClear;
  end;
end;
```

procedure NewImage(I:TImage; F:TColor; IW,IH: Integer);

```
// Löscht ein altes Image und initialisiert ein neues Image
// mit der Farbe F
begin
  with I do begin
    Picture := NIL;
    AutoSize := False;
    Stretch := False;
    if (IW < Screen.Width) then I.Left := (Screen.Width - IW) div 2
    else I.Left := 0;
    if (IH < Screen.Height) then I.Top := (Screen.Height - IH) div 2
    else I.Top := 0;

    Width := IW;
    Height := IH;
    PaintImage(I,F,0,0,IW,IH);
    Picture.Bitmap.PixelFormat := pf24bit;
    AutoSize := True;
  end;
end;
```

procedure StretchImage(Quelle,Ziel: TImage; k: Real);

```
// Proportionale Größenänderung eines Quell-Bildes
// auf ein Ziel-Bild mit dem Änderungsfaktor k
var W,H,W1,H1: Integer;
    v: Real;
begin
  W1 := Quelle.Width;
  H1 := Quelle.Height;
  v := H1 / W1;
  if k <= 0 then k := 1;
  if k > 0 then begin
    W := Round(k * W1);
    H := Round(k * H1);
  end;
  Ziel.Picture := NIL;
  Ziel.AutoSize := False;
  Ziel.Stretch := True;
  Ziel.Width := W;
  Ziel.Height := H;
  Ziel.Canvas.StretchDraw(Rect(0,0,W,H),Quelle.Picture.Bitmap);
  Ziel.AutoSize := True;
  Ziel.Stretch := False;
end;
```

procedure RotateImageRight(Quelle: TImage);

```
// Ein Image-Bitmap um 90° nach rechts drehen
type TMyBild = array[0..0] of TRGBQuad;
var P      : PRGBQuad;
    x,y,w,h : Integer;
    RowOut  : ^TMyBild;
    Bild    : TBitMap;
begin
  Quelle.Picture.Bitmap.PixelFormat := pf32bit;
  H := Quelle.Picture.Bitmap.Width;
  W := Quelle.Picture.Bitmap.Height;
  Bild := TBitMap.Create;
  Bild.PixelFormat := pf32bit;
  Bild.Width := W;
  Bild.Height := H;
  for Y := 0 to (H-1) do begin
    RowOut := Bild.ScanLine[Y];
    P := Quelle.Picture.Bitmap.ScanLine[Quelle.Picture.Bitmap.Height-1];
    inc(P,Y);
    For x := 0 to (W-1) do begin
      RowOut[X] := P^;
      inc(P,H);
    end;
  end;
  Quelle.Picture.Bitmap.Assign(Bild);
  Quelle.Picture.Bitmap.PixelFormat := pf24bit;
  Bild.Free;
end;
```

function MirrorImage(BM: TBitMap; Horiz: Boolean): TBitMap;

```
// horizontale oder vertikale Spiegelung
begin
  Result := TBitMap.Create;
  Result.Width := BM.Width;
  Result.Height := BM.Height;
  if Horiz then
    StretchBlt(Result.Canvas.Handle,0,0,
              Result.Width+1,Result.Height,
              BM.Canvas.Handle,BM.Width,0,
              -BM.Width,BM.Height,srcCopy)
  else
    StretchBlt(Result.Canvas.Handle,0,0,
              Result.Width,Result.Height+1,
              BM.Canvas.Handle,0,BM.Height,
              BM.Width,-(BM.Height),srcCopy);
end;
```

procedure RotateImageLeft(Quelle: TImage);

```
// Ein Image-Bitmap um 90° nach links drehen
type TMyHelp = array[0..0] of TRGBQuad;
var P      : PRGBQuad;
    x,y,w,h : Integer;
    RowOut  : ^TMyHelp;
    help    : TBitMap;
begin
  Quelle.Picture.Bitmap.PixelFormat := pf32bit;
  H := Quelle.Picture.Bitmap.Width;
  W := Quelle.Picture.Bitmap.Height;
  help := TBitMap.Create;
  help.PixelFormat := pf32bit;
  help.Width := W;
  help.Height := H;
  for Y := 0 to (H-1) do begin
    RowOut := help.ScanLine[Y];
    P := Quelle.Picture.Bitmap.ScanLine[Quelle.Picture.Bitmap.Height-1];
    inc(P,Y);
    For x := (W-1) downto 0 do begin
      RowOut[X] := P^;
      inc(P,H);
    end;
  end;
  help := MirrorImage(help,False);
  Quelle.Picture.Bitmap.Assign(help);
  Quelle.Picture.Bitmap.PixelFormat := pf24bit;
  help.Free;
end;
```

procedure PrintImage(I: TImage);

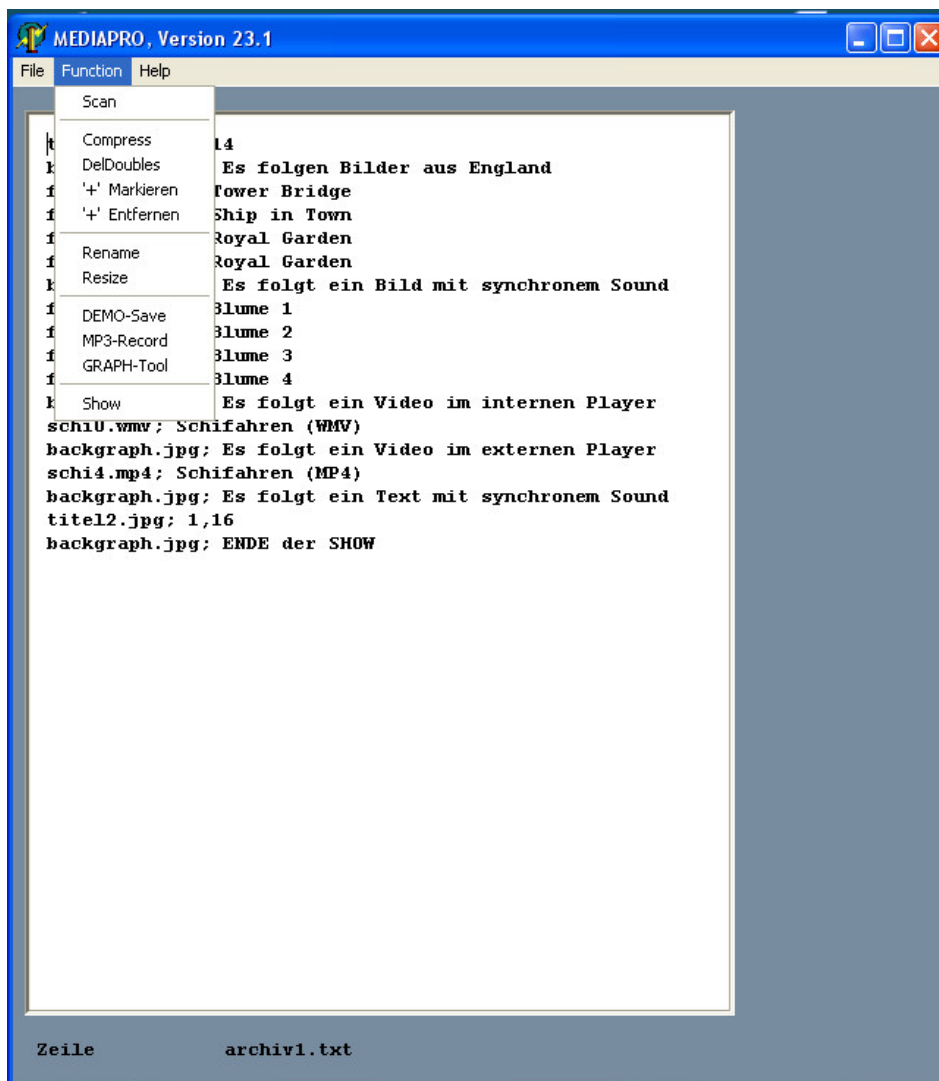
```
// Ausdruck eines Images mit Skalierungsfaktor f (A4-Format mit f = 1)
var Breite, Hoehe: Integer;
    f, Faktor: Double;
    Bereich: TRect;
    S, T: String;
    Code: Integer;
    E: Boolean;
    Bild: TBitmap;
begin
    T := '1.0';
    S := ' A4-skaliertes Ausdruck (20 x 30 cm)';
    E := InputQuery(S, ' Skalierungsfaktor von 0.10 bis 10.0 (0=Abbruch)', T);
    if Not E then Exit;
    val(T, f, Code);
    if (T = '') or (Code > 0) or (f < 0.10) or (f > 10.0) then begin
        MessageBox(0, 'Ausdruck abgebrochen !', 'Problem', 16);
        Exit;
    end;
    with I do begin
        f := abs(f);
        Bild := TBitmap.Create;
        Bild.Width := ClientWidth;
        Bild.Height := ClientHeight;
        Bild.PixelFormat := pf24Bit;
        Bild.Canvas.CopyRect(Rect(0, 0, ClientWidth, ClientHeight),
            Canvas, Rect(0, 0, ClientWidth, ClientHeight));
        Faktor := ClientHeight / ClientWidth;
        Printer.BeginDoc;
        Breite := Round(Printer.Canvas.ClipRect.Right * f);
        Hoehe := Round(Breite * Faktor);
        if Hoehe > Printer.Canvas.ClipRect.Bottom then begin
            Hoehe := Round(Printer.Canvas.ClipRect.Bottom * f);
            Breite := Round(Hoehe / Faktor);
        end;
        Bereich := Rect(0, 0, Breite, Hoehe);
        Printer.Canvas.StretchDraw(Bereich, Bild);
        Printer.EndDoc;
        Bild.Free;
    end;
end;
```


MEDIAPRO

Ein Programm zur Erzeugung von multimedialen Präsentationen.

© Herbert Paukert

Mit dem Programm können schnell und einfach große Foto-Mengen umgeordnet, sortiert, reduziert und dann präsentiert werden. In diese Präsentationen können auch Multimediadateien (Text-, PDF-, MP3-, Videodateien) eingebunden werden. Erfahrungsgemäß soll eine gute Präsentation nicht mehr als ca. 250 Fotos enthalten.



Beschreibung des Programms

Das Programm besteht aus einem Menü zum Ausführen von Aktionen und einem Texteditor zum Anzeigen und Umordnen von Dateinamen.

Die Größe des Programmfensters kann immer durch Ziehen des rechten Fensterrandes verändert werden.



(1) Der Texteditor

- Mit dem Menü **<File>** können Textdateien ("archiv.txt") geöffnet, gedruckt und gespeichert werden.
- Mittels **<File-Demo>** wird eine interne Demo-Show abgespielt. Diese kann mit **<Function-DEMOSave>** in den Ordner "DEMOS" gespeichert werden.

(2) Erfassung von JPG-Bilddateien

- Mittels **<Drag & Drop>** können gewünschte Namen der JPG-Bilddateien aus dem aktuellen Ordner in den Editor transferiert werden, und zwar vor jene Textzeile, auf welcher der Cursor positioniert ist. Dazu sollte das Programm links neben dem Ordner platziert und im Ordner die Dateiansicht auf große Symbole eingestellt werden. Die Liste der erfassten Dateien sollte dann in eine Textdatei „archiv.txt“ abgespeichert werden.

Beim Ablegen der in den Editor mit der Maus gezogenen Dateien wird automatisch an die Dateinamen das Suffix "+" angehängt. So kann man immer erkennen, ob eine Datei bereits im Editor steht. Am Ende müssen mit **<Function '+' Entfernen>** alle Suffixe wieder aus den Dateinamen entfernt werden.

- Mit **<Function-Scan>** werden alle JPG-Bilddateien des aktuellen Ordners erfasst und ihre Namen in der Datei "archiv.txt" gespeichert und im Editor aufgelistet. Dort können sie dann gelöscht oder verschoben werden.

- Mit *<Function-Compress>* werden leere Zeilen und mehrfache Blanks im Editor entfernt.
- Mit *<Function-DelDoubles>* werden mehrfach vorkommende Dateinamen im Editor gelöscht.
- *<Function "+" Markieren>* markiert alle Listen-Dateien mit Suffix "+". Diese Funktion ist nach dem Laden einer Archivdatei sinnvoll, weil man dann die im Editor aufgelisteten Dateien auch im Ordner erkennt.
- *<Function "+" Entfernen>* entfernt von allen Dateien das Suffix "+". Diese Funktion sollte vor dem Speichern der Archivdatei erfolgen.
- Mit *<Function-Rename>* werden alle Dateien der Liste umbenannt.
- Mit *<Function-Resize>* werden alle Dateien der Liste in ihrer Größe verändert und im Unterordner "FILES" gleichnamig abgespeichert.
- Mit *<Function-DEMOSave>* werden alle Dateien der internen Demoshow in den Ordner "DEMOS" abgespeichert.
- Mit *<Function-MP3record>* wird der MP3-Recorder "mp3Directcut.exe" zur Aufnahme von MP3-Sounddateien geöffnet und automatisch im Ordner "TOOLS" gespeichert. Dort kann er dann immer verwendet werden. Die auf diese Weise erzeugten MP3-Sounddateien können in **MEDIAPRO** eingebunden werden.
- Mit *<Function-GRAPHtool>* wird die Grafikverarbeitung "imagine.exe" geöffnet und automatisch im Ordner "TOOLS" gespeichert. Dieses Programm enthält sehr mächtige Funktionen, wie beispielsweise die Stapelumwandlung von Bilddateien von einem Format in ein anderes Format. Wenn Fotodateien in einem anderen Format vorliegen, so können sie auf diese Weise in das JPG-Format umgewandelt und dann in **MEDIAPRO** verwendet werden ("Werkzeuge-Batchkonvertierung").

(3) Darstellung von JPG-Bilddateien

Mit *<Function-Show>* können die im Editor aufgelisteten Bilddateien grafisch dargestellt werden. Dazu gibt es folgende zehn Schalter auf einer verschiebbaren Navigationsleiste:

[>] Bild vorwärts	[D] Bild drucken
[<] Bild rückwärts	[X] Bild löschen
[B] Bildhelligkeit und Kontrast	[?] Hilfstext einblenden
[R] Bild rechts rotieren	[~] Slideshow ein/aus (+Blenden)
[L] Bild links rotieren	[E] Show beenden

Existiert im aktuellen Ordner keine Bilddatei mit dem Dateinamen, dann wird intern ein schwarzes Bild erzeugt. Mit dem Dateinamen "backgraph.jpg" wird intern ein graues Bild erzeugt.

Untertitel werden dadurch erzeugt, indem man sie, durch einen Strichpunkt getrennt, neben die Dateinamen schreibt. Eine optionale Zahl hinter einem weiteren Strichpunkt bestimmt die Schriftgröße des Untertitels.

Existiert im aktuellen Ordner eine Sounddatei "background.mp3", dann wird sie automatisch als Hintergrund-Sound abgespielt.

Existiert im aktuellen Ordner zur Bilddatei "xyz.jpg" eine Sounddatei "xyz.mp3", dann wird sie im internen Mediaplayer synchron zum jeweiligen Bild abgespielt.

Folgende zusätzliche Funktionstasten stehen zur Verfügung:

<F1>	Bildformat (0, 1, 2) einstellen
<Shift F1>	Originalgröße ein/aus
<F2>	Alle Bildeigenschaften einstellen
<Shift F2>	9 Blenden beim Bildwechsel ein/aus (mit 9 Blendengeschwindigkeiten)
<F3>	Slideshow ein/aus
<Shift F3>	Bilderdarbietungszeit ändern
<F4>	optionale Untertitel ein/aus
<Shift F4>	Navigationsleiste ein/aus
<F5>	optionaler Hintergrundsound ein/aus
<F6>	Lautstärke einstellen
<F7>	Sichtbarkeit des internen Mediaplayers ein-/ausschalten, nur bei MP3-Sounddateien oder bei WMV-Videodateien.
<F8>	Externer Mediaplayer ein/aus
<F9>	Sprung zu einem beliebigen Bild

<Strg><T>	Taskleiste einblenden,	<Strg><H>	Taskleiste ausblenden.
-----------	------------------------	-----------	------------------------

In die Zeilen des Editors können auch die Namen von externen Datendateien (Clients) geschrieben werden (PDF-, Sound-, Videodateien). Diese werden automatisch von dem in WINDOWS registrierten Programm (Server) geöffnet. Mittels [>] und [<] wird das Server-Programm beendet und das nächste JPG-Bild dargestellt.

Videodateien im WMV -Format werden standardmäßig im internen Mediaplayer abgespielt, wenn nicht mit <F8> der externe Mediaplayer eingeschaltet ist. Im internen Mediaplayer wird mit Taste <SPACE> ein Videorahmen ein-/ausgeschaltet. Taste <ALT> wechselt zwischen originaler und alternativer (920 x 690)-Videoauflösung. Videodateien, die kein WMV-Format haben, werden automatisch mit dem in WINDOWS registrierten Mediaplayer dargestellt.

(4) Einblenden von mehrzeiligen Texten

- Anlegen einer einfachen Textdatei "titel.txt". Ein Textabschnitt beginnt immer mit einer Zeile #NN, wobei NN die Nummer des Textabschnittes ist. In die letzte Zeile muss immer #0 geschrieben werden.
- Wenn in der Archivdatei "archiv.txt" eine Zeile "titelNN.jpg; p1, p2" steht, dann wird eine leere schwarze Grafik erzeugt und jener Textabschnitt eingeblendet, der in der Textdatei "titel.txt" zwischen den Zeilen #NN und #NN+1 steht.
- Die beiden optionalen Parameter p1 und p2 haben folgende Funktionen: p1 = 0 erzeugt weiße Schrift auf schwarz, und p1 = 1 erzeugt schwarze Schrift auf weiß. Der Parameter p2 bestimmt die Schriftgröße (8 bis 36). In der Standardgröße 16 kann ein Textabschnitt maximal 25 Zeilen enthalten.

Einfaches Beispiel mit "titel.txt"- und "archiv.txt"-Datei:**titel.txt**

#1
Anfang einer Demo-Show
über die Weltstadt Wien
von Herbert Paukert.
#2
Es folgen Bilder aus
der Innenstadt von Wien
mit berühmten Bauten.
#3
Ende der Demo-Show.
#0

archiv.txt

titel1.jpg; 0, 20
foto0001.jpg; Wien und Wienerwald
foto0002.jpg
foto0003.jpg
video.wmv; Am Bisamberg
foto0004.jpg
foto0005.jpg
titel2.jpg; 0, 16
foto0006.jpg; Das Wiener Rathaus
foto0007.jpg
titel3.jpg; 0, 20

(5) Zusätzliche Hinweise

Hinweis 1: Wird der Cursor auf eine Editorzeile (d.h. einem Dateinamen) platziert, dann wird mit der rechten Maustaste die Datei geöffnet.

Hinweis 2: Eine bereits richtig fertiggestellte Archivdatei "archiv.txt" einer Fotoshow sollte sicherheitshalber auch unter "archiv1.txt" zusätzlich abgespeichert werden, um ein Überschreiben zu verhindern.

Hinweis 3: Zusätzlich gibt es vier Notfalltasten: <Strg><F1> = ein Bild vorwärts, <Strg><F2> = ein Bild rückwärts, <Strg><F3> oder <Esc> = eine Show beenden.

Hinweis 4: Mit dem Wechselschalter <F11> wird bestimmt, ob dieser Hilfstext mittels <Help> als Textdatei oder als PDF-Datei angezeigt wird.

Hinweis 5: Das Popup-Menü [~] auf der Navigatorleiste bei der Bildbetrachtung ermöglicht neben dem Ein- und Ausschalten einer automatischen Slideshow noch folgende drei Zusatzfunktionen:

- Einstellen von 3 verschiedenen Bildformaten
- Einstellen von 9 unterschiedlichen Blenden
- Einstellen von 9 Blenden-Geschwindigkeiten

Diese Zusatzfunktionen können auch mit Hilfe der Funktionstasten <F1> und <F2> ausgeführt werden.

Hinweis 6: Für eine schnelle, optimale Bildershow am Computer genügt es, alle großen Fotos mittels <Function Resize> auf das HD-Format (1920 x 1080) zu transformieren.

Auf den nachfolgenden Seiten werden einige, wichtige Routinen des Programmes aufgelistet.

Einige Routinen des Programms

Eine wichtige Funktionalität von **MEDIAPRO** ist, dass in ein **Richedit**-Feld mithilfe von Drag & Drop aus dem WINDOWS-Ordner die Dateinamen der gewünschten JPG-Bilddateien eingelesen werden. Die ausgewählten und sortierten Dateinamen werden dann in einer Textdatei **archiv.txt** gespeichert.

Mit der Prozedur **ShowStart** wird per Schaltknopf die Bildershow gestartet. Dabei werden die Dateinamen in eine Stringliste **MyList1** transferiert, und das erste Bild wird mittels **ShowGraf** in einer Image-Komponente dargestellt, welche in einer **ScrollBar** als Container liegt. Mittels **ShowNext** und **ShowBack** wird dann in der Liste vorwärts oder rückwärts geblättert. Bei jedem Bildwechsel kann der Übergang durch eine Kreuzblende erfolgen, was in der Prozedur **FadeCross** ausgeführt wird. Die globale Variable **Fade1** steuert die Geschwindigkeit der Überblendung. Wenn **Fade1** gleich Null ist, dann ist diese Überblendung ausgeschaltet.

Es können auch die Namen von Videodateien verwendet werden. Falls es sich dabei um WMV- oder MPG-Videos handelt werden sie in einem internen MediaPlayer dargestellt, ansonsten in dem in WINDOWS registrierten MediaPlayer.

Das Programm **MEDIAPRO** enthält die Hauptunit **mediapro_u** und die Nebenunits **ftools_u** und **wtools_u**. In **ftools_u** sind Routinen zur Dateiverwaltung gespeichert, **wtools_u** enthält Routinen zur Taskverwaltung. Nachfolgend werden einige wichtige Routinen beschrieben, welche sich alle in der Hauptunit **mediapro_u** befinden. Diese wird teilweise aufgelistet. Viele Routinen, die durch ein Hauptmenü oder Funktionstasten gesteuert werden, sind hier nicht beschrieben. Im Listing sind nur die hier relevanten visuellen Komponenten und globalen Variablen angeschrieben.

```
procedure TForm1.WMDROPPFILES(var Msg: TMessage);  
// Drag & Drop von externen Dateinamen
```

```
procedure FadeCross(Step, WD, HD: Integer);  
// Crossfading beim Bildwechsel (ausgeschaltet bei Step = 0)
```

```
procedure ShowGraf;  
// Darstellung von JPG-Bilddateien in Image1  
// mit Bildpufferung und Größenanpassung mithilfe von Image2 und Image3
```

```
procedure ShowStart;  
// Die Dateinamen aus RichEdit1 lesen und die Bildershow starten  
// gesteuert mit Schalter [Show]
```

```
procedure ShowNext;  
// ein Bild vorwärts, gesteuert mit Schalter [>]
```

```
procedure ShowBack;  
// Bild rückwärts, gesteuert mit Schalter [<]
```

```
procedure ShowStop;  
// Beenden der Bildershow, gesteuert mit Schalter [E]
```

```
procedure MPlayerON(S: String);  
// Einrichtung eines internen MediaPlayer zum Abspielen  
// von MP3-Sounddateien oder WMV- und MPG-Videodateien
```

```
procedure MPlayerOFF;  
// Ausschalten des internen MediaPlayer
```


procedure TForm1.WMDROFILES(var Msg: TMessage);

```
// Drag & Drop externer Dateinamen
var
  i, count, size: integer;
  filename: PChar;
  s: string;
begin
  inherited;
  count := DragQueryFile(Msg.WParam, $FFFFFFFF, filename, 255);

  for i := 0 to count - 1 do begin
    size := DragQueryFile(Msg.WParam, i, nil, 0) + 1;
    filename := StrAlloc(size);
    try
      DragQueryFile(Msg.WParam, i, filename, size);
      s := Trim(ExtractFileName(StrPas(filename)));
      RichEdit1.Lines.Insert(zei,s);
    finally
      StrDispose(filename);
    end;
  end;

  DragFinish(Msg.WParam);
  Clipboard.Clear;
  RichEdit1.SetFocus;
end;
```

procedure TForm1.FormCreate(Sender: TObject);

```
// Initialisierungen
var
  .....
begin
  Image0 := TJPEGImage.Create;
  Image0.Performance := jpBestQuality;
  Image0.PixelFormat := jf24Bit;
  MyList1 := TStringList.Create;
  .....
  .....

  if Screen.Width > ScrW then begin
    DW0 := ScrW;
    DH0 := ScrH;
  end
  else begin
    DW0 := Screen.Width;
    DH0 := Screen.Height;
  end;
  Image1.Width := DW0;
  Image1.Height := DH0;
  Image2.Width := DW0;
  Image2.Height := DH0;
  imTime := 4000;
  Fade1 := 64;
  Fade2 := Fadel;
  .....
  .....
```

DragAcceptFiles(Handle,True);

```
end;
```

**procedure TForm1.RichEdit1MouseDown(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);**

```
// Aktuelle Zeile lesen und angeklickte Datei ausführen
// (auch Internetadressen "www.xyz" sind hier möglich)
var S,T,V: String;
    H: THandle;
begin
  zeI := RichEdit1.CaretPos.Y;
  if Shift = [ssRight] then begin
    V := GetCurrentDir;
    S := Trim(LowerCase(RichEdit1.Lines[zeI]));
    S := V + '\' + S;
    if FileExists(S) then H := ExecuteFile(S,'','',SW_SHOWNORMAL);
  end;
end;
```


procedure FadeCross(Step, WD, HD: Integer);

```

// Crossfading von Image2 auf Image1
// mit konstanter Bildbreite WD und konstanter Bildhöhe HD
// und mit der Schrittzahl Step (0, 32, 64, 128)

type
  PRGBTripleArray = ^TRGBTripleArray;
  TRGBTripleArray = array[0..32767] of TRGBTriple;

var
  Bitmap1, BitMap2: TBitmap;
  Row1, Row2, Row3: PRGBTripleArray;
  w, h, x, y, n, t: integer;
  c, d: Extended;
  b: Byte;

begin
  if Step = 0 then Exit; // Bildwechsel OHNE Crossfading

  c := Step;
  d := log2(c);
  b := Round(d);
  w := WD;
  h := HD;

  try

    Bitmap1 := TBitmap.Create;
    Bitmap2 := TBitmap.Create;
    Bitmap1.PixelFormat := pf24bit; // or pf32bit
    Bitmap2.PixelFormat := pf24bit;
    Bitmap1.Assign(Form1.Image1.Picture.Bitmap);
    Bitmap2.Assign(Form1.Image2.Picture.Bitmap);

    Form1.Image1.Picture.Bitmap.Width := w;
    Form1.Image1.Picture.Bitmap.Height := h;
    Form1.Image1.Left := (Screen.Width - Form1.Image1.Width) div 2;
    Form1.Image1.Top := (Screen.Height - Form1.Image1.Height) div 2;

    for n := 0 to Step do begin
      for y := 0 to h-1 do begin
        Row1 := Bitmap1.Scanline[y];
        Row2 := Bitmap2.Scanline[y];
        Row3 := Form1.Image1.Picture.Bitmap.Scanline[y];

        for x := 0 to (w - 1) do begin
          Row3[x].rgbtRed := Row1[x].rgbtRed*(Step-n) shr b +
            Row2[x].rgbtRed*n shr b;
          Row3[x].rgbtGreen := Row1[x].rgbtGreen*(Step-n) shr b +
            Row2[x].rgbtGreen*n shr b;
          Row3[x].rgbtBlue := Row1[x].rgbtBlue*(Step-n) shr b +
            Row2[x].rgbtBlue*n shr b;
        end;
      end;

      Form1.Image1.Refresh;
    end;

  finally
    Bitmap1.Free;
    Bitmap2.Free;
  end;
end;

```

procedure PaintImage(I: TImage; F: TColor; X1,Y1,X2,Y2 : Integer);

```
// Füllt ein Image-Rechteck mit einer Farbe
begin
  with I.Canvas do begin
    Pen.Color := F;
    Brush.Color := F;
    Brush.Style := bsSolid;
    FillRect(Rect(X1,Y1,X2,Y2));
    Brush.Style := bsClear;
  end;
end;
```

procedure NewImage(I:TImage; F:TColor; IW,IH: Integer);

```
// Löscht ein altes Image und initialisiert ein neues Image
// mit der Farbe F
begin
  with I do begin
    Picture := NIL;
    AutoSize := False;
    Stretch := False;
    if (IW < Screen.Width) then I.Left := (Screen.Width - IW) div 2
    else I.Left := 0;
    if (IH < Screen.Height) then I.Top := (Screen.Height - IH) div 2
    else I.Top := 0;

    Width := IW;
    Height := IH;
    PaintImage(I,F,0,0,IW,IH);
    Picture.Bitmap.PixelFormat := pf24bit;
    AutoSize := True;
  end;
end;
```

procedure StretchImageInto(Quelle, Ziel: TImage; WD, HD: Integer);

```
// Optimale Einfügung eines Quell-Bildes in einen Teil eines Ziel-Bildes,
// welches die konstante Breite WD und die konstante Höhe HD hat.
```

```
var
  W,H,W1,H1,x,y: Integer;
  v,k: Real;
begin
  W1 := Quelle.Width;
  H1 := Quelle.Height;
  v := H1 / W1;
  k := 1;
  if (W1 <= WD) and (H1 <= HD) then begin
    W := Round(k * W1);
    H := Round(k * H1);
  end;
  if (W1 <= WD) and (H1 > HD) then begin
    H := Round(k * HD);
    W := Round(H / v);
  end;
  if (W1 > WD) and (H1 <= HD) then begin
    W := Round(k * WD);
    H := Round(W * v);
  end;
  if (W1 > WD) and (H1 > HD) then begin
    W := Round(k * WD);
    H := Round(W * v);
    if (H > HD) then begin
      H := Round(k * HD);
      W := Round(H / v);
    end;
  end;
  Ziel.Picture := NIL;
  Ziel.AutoSize := False;
  Ziel.Stretch := True;
  Ziel.Width := WD;
  Ziel.Height := HD;
  PaintImage(Ziel,clBlack,0,0,WD,HD);
  x := abs(WD - W) div 2;
  y := abs(HD - H) div 2;
  Ziel.Canvas.StretchDraw(Rect(x,y,x+W,y+H),Quelle.Picture.Bitmap);
  Ziel.Picture.Bitmap.PixelFormat := pf24bit;
  Ziel.AutoSize := True;
  Ziel.Stretch := False;
end;
```

```
function inVideoTest(S: String): Boolean;
// überprüft das Videoformat eines Videodatei-Namens
begin
  Result := False; inVideo := False; VideoFlag := False;
  if (Pos('.wmv',S) > 0) or (Pos('.mpg',S) > 0) then begin
    Result := True; inVideo := True; VideoFlag := True;
  end;
end;
```

```
function SoundTest(S: String): Boolean;
// überprüft das Soundformat eines Sounddatei-Namens
begin
  Result := False; SoundFlag := False;
  if (Pos('.mp3',S) > 0) then begin
    Result := True; SoundFlag := True;
  end;
end;
```

```
procedure TForm1.Timer2Timer(Sender: TObject);
// MediaPlayer-Position anzeigen, Zeitintervall = 1000 MSec
var z: Double;
begin
  if SoundFlag or inVideo then begin
    if (MedLen > 0) then z := ((100 / MedLen) * Form1.MediaPlayer1.Position);
    Form1.ScrollBar1.Position := Round(abs(z));
    Form1.Label24.Caption := IntToStr(Form1.MediaPlayer1.Position) +
      ':' + IntToStr(MedLen);
    Application.ProcessMessages;
  end;
end;
```

```
procedure TForm1.MediaPlayer1Click(Sender: TObject; Button: TMPBtnType;
var DoDefault: Boolean);
```

```
// Aktivierung der Playertasten
var z: Double;
begin
  with Form1.MediaPlayer1 do begin
    Stop;
    case Button of
      btPlay : Form1.Timer2.Enabled := True;
      btStep : Position := Position + MedFra;
      btBack : Position := Position - MedFra;
      btNext : Position := MedLen - 1;
      btPrev : Position := 0;
    end;
    Form1.Label24.Caption := IntToStr(Form1.MediaPlayer1.Position) + ':' +
      IntToStr(MedLen);
    z := (100 * Form1.MediaPlayer1.Position) / MedLen;
    Form1.ScrollBar1.Position := Round(abs(z));
    Application.ProcessMessages;
  end;
end;
```

```
procedure MPlayerOff;
// Mediaplayer ausschalten
begin
  try
    if NOT Form1.MediaPlayer1.Enabled then Exit;
    if (Form1.MediaPlayer1.Mode <> mpPlaying) and
      (Form1.MediaPlayer1.Mode <> mpStopped) then Exit;
    Form1.Mediaplayer1.AutoEnable := True;
    Form1.MediaPlayer1.Stop;
    Form1.MediaPlayer1.Close;
    Form1.MediaPlayer1.Enabled := False;
    Form1.Timer2.Enabled := False;
    MedLen := 0;
    Form1.Label24.Caption := '0:0';
    Form1.ScrollBar1.Position := 0;
    Form1.Panel3.Visible := False;
    Form1.Panel4.Visible := False;
  except
    end;
end;
```

```

procedure MPlayerON(S: String);
// Mediaplayer einschalten und S abspielen
var w0,h0,w1,h1,x,y: Integer;
    fk: Real;
    Rect0: TRect;
begin
  if NOT FileExists(S) then Exit;
  if SoundFlag and NOT VideoFlag then begin
    try
      Form1.MediaPlayer1.Enabled := True;
      Form1.MediaPlayer1.AutoEnable := False;
      Form1.MediaPlayer1.FileName := S;
      Form1.MediaPlayer1.Open;
      Form1.MediaPlayer1.Wait := False;
      Form1.MediaPlayer1.Notify := False;
      MedLen := Form1.MediaPlayer1.Length;
      if (MedLen = 0) then begin MPlayerOff; Exit; end;
      Form1.MediaPlayer1.Frames := MedLen div 10;
      MedFra := Form1.MediaPlayer1.Frames;
      MedPos := 0;
      Form1.Timer2.Interval := 1000;
      Form1.Label24.Caption := '0:0';
      Form1.Label24.Visible := True;
      Form1.Panel3.Visible := True;
      if NOT SlideFlag then Form1.Timer2.Enabled := True;
      Form1.MediaPlayer1.Play;
    except
      MPlayerOff;
    end;
  end;
  if VideoFlag and inVideo then begin
    try
      Form1.MediaPlayer1.Enabled := True;
      Form1.MediaPlayer1.AutoEnable := False;
      Form1.MediaPlayer1.FileName := S;
      Form1.MediaPlayer1.Open;
      Form1.MediaPlayer1.Wait := False;
      Form1.MediaPlayer1.Notify := False;
      MedLen := Form1.MediaPlayer1.Length;
      if (MedLen = 0) then begin MPlayerOff; Exit; end;
      Form1.MediaPlayer1.Display := Nil;
      Rect0 := Form1.MediaPlayer1.DisplayRect;
      w0 := (Rect0.Right - Rect0.Left);
      h0 := (Rect0.Bottom - Rect0.Top);
      if inVideoSi = 1 then begin
        w1 := w0; h1 := h0;
      end;
      if inVideoSi = 2 then begin
        w1 := 920;
        fk := w1 / w0;
        h1 := Round(h0*fk);
      end;
      Form1.Panel4.Width := w1;
      Form1.Panel4.Height := h1;
      x := 0;
      y := 0;
      if (w1 < Screen.Width) then x := (Screen.Width - w1) div 2;
      if (h1 < Screen.Height) then y := (Screen.Height - h1) div 2;
      Form1.Panel4.Left := x;
      Form1.Panel4.Top := y;
      if inVideoBo then Form1.Panel4.BorderStyle := bsSingle
        else Form1.Panel4.BorderStyle := bsNone;
      Form1.Panel4.BringToFront;
      Form1.Panel4.Visible := True;
      Form1.MediaPlayer1.Display := Form1.Panel4;
      Form1.MediaPlayer1.DisplayRect := Rect(0,0,w1,h1);
      Form1.MediaPlayer1.Frames := MedLen div 10;
      MedFra := Form1.MediaPlayer1.Frames;
      MedPos := 0;
      Form1.Timer2.Interval := 1000;
      Form1.Label24.Caption := '0:0';
      Form1.Label24.Visible := True;
      Form1.Panel3.Visible := True;
      if NOT SlideFlag then Form1.Timer2.Enabled := True;
      Form1.MediaPlayer1.Play;
    except
      MPlayerOff;
    end;
  end;
end;
end;

```

procedure ShowGraf;

```

// Kernprozedur zur Darstellung einer JPG-Bilddatei
// in einer Image-Komponente Image1
var
  S: String;
  VProg: String;
  .....
begin
  S := MyList1[imCount]; // Dateinamen holen
  S := Trim(LowerCase(S));

  if VideoFlag and NOT inVideo then begin // externes Videoprogramm
    VProg := GetExeForFile(VName); // und Soundprogramm
    KillTask(VProg); // beenden
  end;
  if SoundFlag or VideoFlag then begin
    SoundFlag := False;
    VideoFlag := False;
    inVideo := False;
    MPlayerOFF;
  end;
  VName := '';
  SName := '';
  VProg := '';

  if SoundTest(S) then begin // MP3-Sounddatei
    NewImage(Form1.Image1,clBlack,1000,750); // im internem Player
    Fade1 := 0; // abspielen
    MPlayerOff;
    SName := S;
    MPlayerON(SName);
    Exit;
  end;

  P := Pos('.jpg',S);
  if (P = 0) then begin
    NewImage(Form1.Image1,clBlack,1000,750);
    Fade1 := 0;
    MPlayerOFF;
    VName := S; // Video
    if inVideoTest(VName) then MPlayerON(VName) // im internen Player
abspielen
    else begin
      H := ExecuteFile(VName,"",SW_SHOWNORMAL); // im externen Player
abspielen
      if Integer(H) < 32 then ShowMessage('"' + S + '" nicht gefunden !');
    end;
    Exit;
  end;

  .....
  .....

  GName := S;
  Image0.PixelFormat := jf24Bit;
  Image0.Performance := jpBestQuality;
  Image0.LoadFromFile(GName); // JPG-Bilddatei laden

  Image3.AutoSize := True;
  Image3.Stretch := False;
  Image3.Picture.Bitmap.Assign(Image0); // Bildpufferung
  Image2.AutoSize := False;
  Image2.Stretch := True;
  Image2.Width := DW0;
  Image2.Height := DH0;
  StretchImageInto(Form1.Image3, Form1.Image2, DW0, DH0); // Größenanpassung
  Image1.AutoSize := False;
  Image1.Stretch := True;
  Image1.Width := Image2.Width;
  Image1.Height := Image2.Height;
  Image1.AutoSize := True;
  Image1.Stretch := False;
  FadeCross(Fade1, DW0, DH0); // CrossFading
  // ausgeschaltet bei Fade1

= 0
Image1.Picture.Bitmap.Assign(Form1.Image2.Picture.BitMap); // Bilddarstellung
Image1.Left := (Screen.Width - Image1.Width) div 2;
Image1.Top := (Screen.Height - Image1.Height) div 2;
Fade1 := Fade2;
end;

```

procedure ShowStart;

```
// Die Dateinamen aus RichEdit1 lesen und die Bildershow starten
// gesteuert vom Schalter [Show]
var
    .....
begin
    .....
    .....
    imMax := Form1.RichEdit1.Lines.Count;
    MyList1.Clear;
    For I := 0 to imMax-1 do begin
        S := Trim(RichEdit1.Lines[I]);
        P := Pos(';',S);
        if P > 0 then S := Trim(Copy(S,1,P-1));
        MyList1.Add(S);
    end;
    imCount := 0;

    FW := Form1.Width;
    FH := Form1.Height;
    RichEdit1.Visible := False;
    Form1.Align := alClient;
    Form1.Color := clBlack;
    ScrollBox1.Align := alClient;
    ScrollBox1.Color := clBlack;
    ScrollBox1.Visible := True;
    Image1.Visible := True;
    SlideFlag := False;
    ShowFlag := True;
```

ShowGraf;

```
end;
```

procedure ShowNext;

```
// ein Bild vorwärts, gesteuert vom Schalter [>]
begin
    imCount := imCount + 1;
    if imCount >= imMax then imCount := 0;
    ShowGraf;
end;
```

procedure ShowBack;

```
// Bild rückwärts, gesteuert vom Schalter [<]
begin
    imCount := imCount - 1;
    if imCount <= -1 then imCount := imMax-1;
    ShowGraf;
end;
```

procedure ShowStop;

```
// Bildershow beenden, gesteuert mit Schalter [E]
begin
    with Form1 do begin
        MPlayerOFF;
        SlideFlag := False;
        ShowFlag := False;
        MyList1.Clear;
        NewImage(Form1.Image1,clBlack,1000,750);
        ScrollBox1.Align := alNone;
        ScrollBox1.Visible := False;
        Form1.Align := alNone;
        Form1.AutoSize := False;
        Form1.AutoScroll := True;
        Form1.BorderStyle := bsSizeable;
        Form1.BorderWidth := 0;
        Form1.Left := 0;
        Form1.Top := 0;
        Form1.Width := FW;
        Form1.Height := FH;
        Form1.Color := FCol;
        RichEdit1.Visible := True;
        .....
        .....
    end;
end;
```

procedure SlideShow;

```
// Slideshow ein/aus, gesteuert mit Wechselschalter [~]
var S,T: String;
    Code: Integer;
begin
  SlideFlag := NOT SlideFlag;
  if SlideFlag then begin
    S := '4';
    T := 'Automatische Slideshow steuern' + #13 +
        'mit verschiebbarer Navigatorleiste.' + #13 +
        ' ';
    S := InputBox('Darbietungszeit (1 bis 10 Sekunden)',T,S);
    Val(S,imTime,Code);
    if (Code <> 0) or (imTime < 1) or (imTime > 10) then imTime := 4;
    imTime := 1000 * imTime;
    GName := MyList1[imCount];
    Form1.Timer1.Interval := imTime;
    Form1.Timer1.Enabled := True;
    ShowGraf;
    Exit;
  end;
  if NOT SlideFlag then begin
    Form1.Timer1.Enabled := False;
    ShowMessage('Slideshow angehalten .....');
  end;
end;
```

procedure TForm1.Timer1Timer(Sender: TObject);

```
// Automatischer Bildwechsel bei Slideshow, Zeitintervall = imTime
begin
  if NOT SlideFlag then begin Timer1.Enabled := False; Exit; end;
  if VideoFlag and NOT inVideo then Exit;
  if (SoundFlag or inVideo) and (Form1.MediaPlayer1.Mode = mpPlaying) then Exit;
  imCount := imCount + 1;
  if (imCount >= imMax) then imCount := 0;
  ShowGraf;
end;
```

procedure TForm1.ScrollBar1Enter(Sender: TObject);

```
// Scrollbar anklicken zur Steuerung des internen Mediaplayers
begin
  if NOT Form1.ScrollBar1.Visible then Exit;
  Form1.Timer2.Enabled := False;
  if Form1.MediaPlayer1.Mode = mpPlaying then Form1.MediaPlayer1.Pause;
end;
```

**procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
var ScrollPos: Integer);**

```
// Scrollbar scrollen zur Steuerung des internen Mediaplayers
var z: Double;
    p: Int64;
begin
  if NOT Form1.ScrollBar1.Visible then Exit;
  if SoundFlag or inVideo then begin
    Form1.Timer2.Enabled := False;
    if Form1.MediaPlayer1.Mode = mpPlaying then Form1.MediaPlayer1.Pause;
    z := (ScrollPos * MedLen) / 100;
    p := Round(abs(z));
    if p >= MedLen then p := MedLen;
    Form1.MediaPlayer1.Position := p;
    Form1.Label24.Caption := IntToStr(p) + ':' + IntToStr(MedLen);
    Application.ProcessMessages;
  end;
end;
```

----- E N D E -----