VFD-STUDIO

Das Hauptprogramm
Dokumentation der Unit MainUnit.pas

Inhalt

- *Vorwort:*

Das VFD-Studio besteht u.a. aus drei ausführbaren Dateien (*.exe): Setup.exe, ListCreator.exe und VFDStudio.exe
`Setup.exe` ist ein kleines Programm zur Einstellung verschiedener Werte der VFD.ini, auf die VFDStudio.exe zugreift. Es wird als letztes beschrieben.
`ListCreator.exe` dient der Erstellung von „Screen-Listen“ (*.lst). Diese Screen-Listen enthalten Anweisungen, welche VFDStudio.exe interpretiert und auf dem Display ausgibt.
Diese Listen-Scriptsprache wird in einem eigenen Kapitel behandelt.
`VFDStudio.exe` ist das eigentliche Programm, welches das Display anhand der Befehle in den Listen ansteuert und wird folgend näher beschrieben.

Die *MainUnit.pas* hat mehrere Funktionen, die sich grob in zwei Bereiche aufteilen lassen: Benutzeroberfläche und Steuerung des Datenflusses.

Die Benutzeroberfläche ist im wesentlichen durch das Hauptfenster *MainForm*, sowie dessen visuelle Komponenten gegeben.
Sie gibt dem Anwender die Möglichkeit in den Programmablauf einzugreifen, um beispielsweise eine andere Liste zu laden.

Die Steuerung des Datenflusses wird hingegen von den nicht-visuellen Komponenten bestimmt.
Als „Datenfluss“ ist dabei das Interpretieren der Befehle der Listen, das Ersetzen bestimmter Schlüsselwörter durch Systeminformationen und schließlich die Ausgabe der Informationen auf dem Display zu verstehen.

Übersicht der globalen Variablen der MainUnit:

`MainForm: TMainForm;`
MainForm bezeichnet das Hauptfenster – die Benutzeroberfläche

`PosX, PosY: integer;`
Diese Position gibt die Stelle auf dem Bildschirm an, an der das Popupmenü aufklappt, wenn der Benutzer auf das VFD-Studio-Symbol in der Taskleiste rechtsklickt.

`ListIndex: integer;`
Die auszuführende Liste mit Anweisungen ist in Blöcke aufgeteilt (Screens). Listindex gibt die aktuell bearbeitete Zeile innerhalb dieses Blockes an.

`InfoStrings: Array[0..7] of InfoString;`
Es wird unterschieden zwischen Informationen, die nicht ständig aktualisiert werden müssen (z.B. Name des Betriebssystemes) und Informationen, die immer wieder neu angezeigt werden müssen (z.B. Uhrzeit).
Das Display kann acht Zeilen Text darstellen; enthält eine dieser Zeilen Informationen, die ständig aktualisiert werden müssen, so wird diese Zeile in das InfoStrings-Array übertragen und dann sekündlich auf dem Display ausgegeben.
InfoString ist ein Record folgenden Types:

```
type
  InfoString = record
   Info: string;      // enthält auszugebenden Text mit Schlüsselwörtern
   X,Y: byte;         // Spalte und Zeile auf dem Display
  end;
```

`Fonted: Boolean;`
Legt fest ob die Textausgabe in Schriftart (true) oder normal (false) erfolgen soll.

`CPUUsage: Array[0..449]of single;`
Dieses Array enthält die letzten 450 Messungen der CPU-Last, um daraus einen Durchschnittswert zu bestimmen. Dies entspricht einem Zeitraum von 1000 Sek. ~ 14 Minuten.

`Cu, Mu: real;`
Cu enthält die aktuelle CPU-Last, Mu die aktuelle Speichernutzung. Beide Angaben in Prozent.

`RAMGraph: Array[0..252] of Byte;`
RAMGraph speichert die letzten die letzten 253 Werte der Speichernutzung. Angaben in Prozent ohne Kommastellen.

```
CPUMONITORenabled: Boolean;
CPUMONITORcharEnabled: Boolean;
RAMMONITORenabled: Boolean;
RAMMONITORcharEnabled: Boolean;
```

Es gibt sechs Extrascreens. Dies sind vorgefertigte Screens, die nur über einen einzelnen Befehl aktiviert werden müssen.
Obige Variablen geben an, ob einer der „Monitor“-Extrascreens aktiviert ist.

`ScreenAnimated: Bool;`
Diese Variable ist true, wenn der aktuelle Screen eine Animation enthält.

`PlayOnlyOnIdle: Bool;`
Diese Variable gibt an, ob Animationen nur abgespielt werden, wenn die durchschnittliche CPU-Last idle ist.

`IdleLevel: Integer;`
IdleLevel gibt an, welche durchschnittliche CPU-Last noch als idle gilt. Angabe in Prozent.

`IdleTime: Integer;`
Wie lange muss die durchschnittliche CPU-Last idle gewesen sein?

`AnimationPaused: Bool;`
Gibt an, ob das Abspielen von Animationen pausiert wurde.

`Drives: String;`
Dieser String ist eine Liste der installierten Laufwerke (z.B. 'ACDE')

`DriveIndex: Integer;`
Gibt das aktuelle Laufwerk an (entspricht der Position in obigen String)

Das Hauptfenster MainForm:

Wie oben beschrieben ist MainForm vom Typ TMainForm. Nachfolgend nun auch hierzu eine Übersicht:

```
type
  TMainForm = class(TForm)
    PopupMenu: TPopupMenu;           //Popupmenü, für Steuerung durch Symbol in Taskleiste
    Exit1: TMenuItem;                //Menüpunkt „Exit“ des Popupmenüs
    Show1: TMenuItem;                //Menüpunkt des Popupmenüs um das Hauptfenster anzuzeigen
    HideTimer: TTimer;               //TimerObjekt HideTimer – siehe Prozedur HideTimerTimer
    ExitButton: TBitBtn;             //Schaltfläche auf dem Hauptfenster für Programmende
    OKButton: TBitBtn;               //Schaltfläche um Hauptfenster in SysTray zu minimieren
    DLPortIO: TDLPortIO;             //TDLPortio-Komponente – siehe TPortIOVFD-Dokumentation
    N1: TMenuItem;
    N2: TMenuItem;                   //N1 und N2 dienen nur der optischen Aufteilung des Popupmenüs
    VFD: TPortIOVFD;                 //TPortIOVFD-Komponente
    ListTestButton: TButton;         //Button “Liste laden…”
    ListBox: TListBox;               //Diese Listbox enthält die List-Datei
    WaitTimer: TTimer;               //TimerObjekt WaitTimer – siehe Prozedur WaitTimerTimer
    ExtListBox: TListBox;            //Lädt weitere Listen, um sie der aktuellen hinzuzufügen
    SysInfo: TSysInfo;               //Komponente zur Informationsbeschaffung (s. TSysInfo-Dokumentation)
    InfoTimer: TTimer;               //TimerObjekt InfoTimer – siehe Prozedur InfoTimerTimer
    FontLabel: TLabel;               //Dient lediglich als Platzhalter für TFont-Eigenschaften
    Winamp: TWinampControl;          //Komponente zur Ermittlung von Winamp-Informationen
    CPU: Tacpuid;                    //Komponente zur Ermittlung von CPU-Informationen
    NextButton: TButton;             //Schaltfläche “Nächster Screen >>“
    StopButton: TButton;             //Schaltfläche “Stop”
    SCR_Time_Label: TLabel;          //Label zur Anzeige der Screen-Anzeigedauer
    OpenDialog: TOpenDialog;         //Dialog zum öffnen von Listendateien
    ViewListPanel: TPanel;           //Klick auf diese Panel für erweiterte/normale Ansicht
    Bevel2: TBevel;
    Bevel1: TBevel;
    Bevel3: TBevel;
    Bevel4: TBevel;                  //Die Bevels dienen nur der optischen Aufteilung der MainForm
    Image1: TImage;                  //Imageobjekt zur Anzeige des VFD-Studio Logos
    ExtraTimer: TTimer;              //TimerObjekt ExtraTimer siehe Prozedur ExtraTimerTimer
    UsageTimer: TTimer;              //TimerObjekt UsageTimer – siehe Prozedur UsageTimerTimer
    PopupStopButton: TMenuItem;      //Menüpunkt “Stop” des Popupmenüs
    NchsterScreen1: TMenuItem;       //Menüpunkt “Nächster Screen >>” des Popupmenüs
    Listeladen1: TMenuItem;          //Menüpunkt “Liste laden..." des Popupmenüs
    InfoButton: TBitBtn;             //Schaltfläche für Infofenster
    OnlyOnIdleBox: TCheckBox;        //Checkbox „Animationen werden nur abgespielt wenn CPU idle ist"
    procedure Exit1Click(Sender: TObject);          //Beendet das Programm
    procedure Show1Click(Sender: TObject);          //Zeigt das Haupfenster an
    procedure HideTimerTimer(Sender: TObject);      //Initialisiert das Display und lädt letzte Liste
    procedure ExitButtonClick(Sender: TObject);     //Beendet das Programm
    procedure OKButtonClick(Sender: TObject);       //Minimiert das Hauptfenster
    procedure FormCreate(Sender: TObject);          //liest Ini-Datei, initialisiert vfd-Komponente,...
    procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);  //Beendet das Programm
    procedure ListTestButtonClick(Sender: TObject); //zeigt Dialogfenster für Liste aus Datei laden
    procedure WaitTimerTimer(Sender: TObject);      //erstellt nächsten Screen nach Ablauf des aktuellen
    procedure InfoTimerTimer(Sender: TObject);      //aktualisiert CPU- und Memory-Auslastung
    procedure FormDestroy(Sender: TObject);
    procedure NextButtonClick(Sender: TObject);     //zeigt nächsten Screen an
    procedure StopButtonClick(Sender: TObject);     //hält WaitTimer an
    procedure ViewListPanelClick(Sender: TObject);  //zeigt erweiterte/normale Ansicht d. Hauptfensters
    procedure ExtraTimerTimer(Sender: TObject);     //zeigt Laufwerksinformationen an
    procedure UsageTimerTimer(Sender: TObject);     //aktualisiert CPU- und Memory-Nutzungsarray
    procedure PopupStopButtonClick(Sender: TObject);//hält Waittimer an
    procedure NchsterScreen1Click(Sender: TObject); //zeigt nächsten Screen an
    procedure Listeladen1Click(Sender: TObject);    //zeigt Dialogfenster für Liste aus Datei laden
    procedure InfoButtonClick(Sender: TObject);     //zeigt Infofenster an
    procedure OnlyOnIdleBoxClick(Sender: TObject);  //schaltet PlayOnlyOnIdle zwischen true/false
    procedure OnlyOnIdleBoxKeyPress(Sender: TObject; var Key: Char); //wie OnlyOnIdleBoxClick
  private
    IconData: TNewNotifyIconData;                   //für das SysTray-Icon
    procedure WMSysCommand(var Message: TWMSysCommand); message WM_SysCommand;   //sendet Message
    procedure SysTrayIconMsgHandler(var Msg: TMessage); message TRAY_CALLBACK;
    procedure AddSysTrayIcon;                       //Fügt Icon in SystemTray hinzu
    procedure ShowBalloonTips;                      //Zeigt Sprechblase
    procedure DeleteSysTrayIcon;                    //Entfernt SysTray-Icon
    procedure Interprete(s:String);                 //Interpretiert den angegebenen String
    procedure CreateScreen;                         //erstellt einen neuen Screen
    procedure BitmapToVFD(fname: string; XDisp,YDisp,Wdth, XOff: Word; Farbe: TColor); //Zeigt Bild auf VFD
    procedure LoadList(ListFileName: string);       //Lädt Liste aus Datei (und fügt weitere Listen ein)
    procedure InfoString(s: string; x,y: byte);     //ersetzt Schlüsselwörter durch Systeminformationen
    procedure ClearInfoStrings;                     //löscht alle Einträge des InfoString-Arrays
```

```
  procedure RefreshInfos;                        //ersetzt Schlüsselwörter durch Systeminformationen
  procedure CharBar(Xa,Xe,Line:Byte;Percent:double); //Zeigt eine %-Bar an (für Laufwerksinformationen)
  procedure showCPUMONITOR;                      //Diese vier Prozeduren
  procedure showCPUMONITORchar;                    //zeichnen einen Rahmen
  procedure showRAMMONITOR;                        //für den jeweiligen ExtraScreen
  procedure showRAMMONITORchar;                    //auf das Display.
  procedure showPrinterScreen;                   //Zeigt den ExtraScreen der Druckerübersicht
  procedure showDrivesScreen;                    //Zeigt ExtraScreen mit Laufwerksinformationen
  procedure ShowExtraScreens;                    //Aktualisierung der ExtraScreens
  function averageCPUusage: single;              //Berechnet durchschnittliche CPU-Last
  function isIdle: Bool;                         //Prüft, ob die CPU idle ist
public
  //Einstellungen aus der IniDatei
  LPTPort: word;                       // Portadresse
  ToDoList: string;                    // ListName
  //Für Extratimer
  ShowTime,OverviewTime: integer;      //Anzeigedauer der einzelnen Laufwerke und der Übersicht
end;
```

Weiterhin sollte an dieser Stelle noch die Verwendung der Drives.dll erwähnt werden. Deren Funktionen werden wie folgt eingebunden:

```
implementation

{$R *.DFM}

 //Die Funktionen aus der Drives.dll einbinden:
function GetFreeDiskSpace(Drive: char): double; external 'Drives.dll';
function GetDiskSpace(Drive: char): double; external 'Drives.dll';
```

Die Methoden der Unit MainUnit und von MainForm:

Die Methoden dieser Unit sollen nun anhand eines Beispieles erklärt werden.
Dieses Beispiel soll ein mögliches Anwendungsszenario durchspielen um dabei den Zweck der einzelnen Funktionen und Prozeduren zu verdeutlichen.

Dies sind die Stationen des Szenarios aus Sicht des Anwenders:

Anwendungsszenario:

1. Anwender startet das VFD-Studio, VFD-Symbol mit Sprechblase erscheint in der Taskleiste
2. Anwender klickt auf das Symbol und öffnet Menü. Anwender wählt Menüpunkt „Hauptfenster“
3. Anwender wechselt zur erweiterten Ansicht
4. Es wird eine andere Liste geladen
5. Klick auf Button „Nächster Screen >>“
6. Klick auf Button „Stop“
7. Klick auf Button „Go“
8. Klick auf Button „OK“
9. Anwender klickt auf das Symbol und öffnet Menü. Anwender wählt Menüpunkt „Exit“

1. Anwender startet das VFD-Studio, VFD-Symbol mit Sprechblase erscheint in der Taskleiste

Als eine der ersten Methoden wird FormCreate gestartet.
FormCreate füllt zunächst das CPUusage-Array mit *–1*.
Dieses Array speichert die CPU-Last Werte der letzten 15 Minuten. Beim Berechnen des Durchschnittswertes werden dann alle Werte ungleich *–1* zusammengezählt und durch die Anzahl der gültigen Werte geteilt. Auf diese Art und Weise wird in den ersten 15 Minuten der Durchschnitt seit Programmstart berechnet, danach der Durchschnitt der letzten 15 Minuten.
Würde das Array nicht so initialisiert werden, müsste man andernfalls 15 Minuten ab Programmstart warten, bis der Durchschnittswert korrekt berechnet werden kann.

Als nächstes wird durch die Prozedur AddSysTrayIcon ein SysTray-Icon in der Taskbar installiert. D.h. es erscheint ein VFD-Studio-Symbol links neben der Uhr.

Anschließend werden die Konfigurationseinstellungen aus der Vfd.ini Inidatei ausgelesen. Existiert die Datei nicht, wird sie automatisch mit Standardwerten erstellt.
Folgende Werte werden aus der Inidatei ausgelesen und in globalen Variablen gespeichert:

- ShowTime – gibt an wie lange (Angabe in ms) Informationen zu jedem einzelnen Laufwerk angezeigt werden (Default 3 Sek.)
- OverviewTime - gibt an wie lange (Angabe in ms) die Übersicht über alle Laufwerke angezeigt werden soll (Default 6 Sek.)
- Port – gibt die Basisadresse des verwendeten LPTs an. Die Angabe ist in Dezimal. (Default 888 = 378_h)
- Listname – enthält den Dateinamen der zuletzt geöffneten Liste. Diese Liste wird beim erneuten Start automatisch wieder geöffnet.
- PlayOnlyOnIdle – bestimmt, wie der Name schon andeutet, ob Animationen nur abgespielt werden, wenn die CPU idle ist. (Default false)
- IdleTime – gibt an über wie viele Messungen die CPU-Last idle gewesen sein muss (Default 45 Messungen – entspricht 90 Sek.)
- IdleLevel – gibt an, welche CPU-Last noch als idle gilt. (Default 25%)

Schließlich wird der TPortIOVFD-Komponente vfd, welche das Display steuert, eine Referenz auf die Treiber-Komponente DLPortIO gegeben, damit sie direkt mit dieser kommunizieren kann.
Danach wird noch die Position der Treiberdateien angegeben. Diese sollten im Hauptverzeichnis liegen und werden beim ersten Start automatisch in das Windows Setup32-Verzeichnis installiert.

Anschließend wird der vfd-Komponente noch die Adresse des LPT-Ports übermittelt, die IniDatei wieder freigegeben, die Checkbox „Animationen werden nur abgespielt wenn CPU idle ist" gesetzt und eine „Sprechblase“ angezeigt. Diese erscheint über dem VFD-Symbol in der Taskleiste und blendet den Dateinamen der aktuellen Liste ein.
Zum Schluss wird versucht den Treiber zu starten. Misslingt dies wird eine entsprechende Fehlermeldung angezeigt.

```
procedure TMainForm.FormCreate(Sender: TObject);
var
  VFDIni: TIniFile;
  IniName: string;
  i: integer;
begin
    //CPUusage initialisieren - '-1' wird als nicht gesetzter Wert interpretiert
    for i:=0 to 449 do
     cpuusage[i]:=-1;
    width:=260; //Fensterbreite 260 Pixel – entspricht normaler Ansicht
    try
    AddSysTrayIcon;
    IniName:=extractfilepath(application.ExeName)+'Vfd.ini';
    VFDIni := TIniFile.Create(IniName);
    with VFDIni do begin
       mainform.ShowTime:=ReadInteger('EXTRAS', 'ShowTime', 3000);
       mainform.OverviewTime:=ReadInteger('EXTRAS', 'OverviewTime', 6000);
       mainform.LPTPort:=ReadInteger('SETTINGS', 'Port', 888);
       mainform.ToDoList:=ReadString('LIST', 'Listname', '');
       PlayOnlyOnIdle:=ReadBool('ANIMATIONS', 'PlayOnlyOnIdle', false);
       IdleTime:=ReadInteger('ANIMATIONS', 'IdleTime', 45);
       IdleLevel:=ReadInteger('ANIMATIONS', 'IdleLevel', 25);
    end;
    vfd.PortIOComponent:=Dlportio;  //PortIO-Komponente für VFD-Komponente zuweisen
    dlportio.DriverPath:=extractfilepath(application.exename);
    vfd.LPT:=mainform.LPTPort;           //LPT zuweisen
    VFDIni.Free;
    ShowBalloonTips;
    except
    end;
    OnlyOnIdleBox.Checked:=PlayOnlyOnIdle;
    try
     // Treiber öffnen
     DLPortIO.OpenDriver();
     if (not DLPortIO.ActiveHW) then
     begin
        MessageDlg('Konnte DriverLINX-Treiber nicht öffnen', mtError, [mbOK], 0);
     end;
     except
    end;
end;
```

```
// AddSysTrayIcon fügt ein SysTrayIcon hinzu
procedure TMainForm.AddSysTrayIcon;
begin
  try
  IconData.cbSize := SizeOf(IconData);
  IconData.Wnd := AllocateHWnd(SysTrayIconMsgHandler);
  IconData.uID := 0;
  IconData.uFlags := NIF_ICON or NIF_MESSAGE or NIF_TIP;
  IconData.uCallbackMessage := TRAY_CALLBACK;
  IconData.hIcon := Application.Icon.Handle;    //IconHandle
  // Beschreibung, wenn man die Maus auf das Symbol bewegt
  IconData.szTip:='VFD-Studio';
  Shell_NotifyIcon(NIM_ADD, @IconData);  //Icon hinzufügen
  except
  end;
end;
```

ShowBallonTips zeigt beim Starten des VFD-Studios eine Sprechblase an, in welcher der Dateiname (in Variable TodoList) steht.

```
{ShowBalloonTips zeigt die Sprechblase}
procedure TMainForm.ShowBalloonTips;
var
  TipInfo, TipTitle: string;
begin
  try
  IconData.cbSize := SizeOf(IconData);
  IconData.uFlags := NIF_INFO;

  TipInfo:='Liste: '+TodoList;
  strPLCopy(IconData.szInfo, TipInfo, SizeOf(IconData.szInfo) - 1);
  IconData.DUMMYUNIONNAME.uTimeout := 500;
  TipTitle := 'VFD-Studio';
  strPLCopy(IconData.szInfoTitle, TipTitle, SizeOf(IconData.szInfoTitle) - 1);
  IconData.dwInfoFlags := NIIF_INFO;     //NIIF_ERROR;  //NIIF_WARNING;
  Shell_NotifyIcon(NIM_MODIFY, @IconData);
  IconData.DUMMYUNIONNAME.uVersion := NOTIFYICON_VERSION;
  Shell_NotifyIcon(NIM_ADD, @IconData);
  except
  end;
end;
```

Bald nach der Prozedur FormCreate wird das OnTimer-Event des HideTimer aktiviert, dessen Intervall bei nur 1 ms liegt und folgende Prozedur aktiviert:

```
procedure TMainForm.HideTimerTimer(Sender: TObject);
begin
     Hide;
     vfd.InitVFD;
     vfd.ClearScreen(0);
     vfd.ClearScreen(1);
     //Liste laden
     Mainform.LoadList(extractfilepath(application.ExeName)+'Lists\'+TodoList);
     HideTimer.enabled:=false; //Disabeld sich selbst
end;
```

Seine Aufgabe sind das Hauptfenster zu verbergen, die Initialisierung des Displays aufzurufen, die Displayanzeige zu löschen und die LoadList-Prozedur zu starten.
Anschließend schaltet er sich selbst ab und wird nicht länger benötigt.
Man könnte nun einwenden, dafür sei kein Timer erforderlich gewesen sein, sondern diese Aufrufe hätten auch noch in FormCreate stattfinden können – tatsächlich aber führt dies jedoch zu einer Exception, weshalb dieser Weg gegangen werden musste.
Die Ursache des Fehlers ließ sich leider nur auf einen Konflikt mit dem SysTray-Icon zurückführen und nicht genauer bestimmen.

Zu den Methoden InitVFD und ClearScreen sei auf die Dokumentation TPortIOVFD verwiesen.

Der HideTimer ruft die Prozedur LoadList mit dem Dateinamen der zuletzt geöffneten Liste als Parameter auf:

```
procedure TMainForm.LoadList(ListFileName: TFileName);
var
   i: integer;
   s: string;
   hs: string; //Hilfsstring
   z: integer; //Hilfszähler
begin
     //Liste laden
     listbox.Items.LoadFromFile(ListFileName);
     //externe Listen einfügen
     i:=0;
     while i<= listbox.items.count-1 do begin
      s:=listbox.items[i];
      if pos('LOADSCREEN',s)=1 then begin
       hs:=copy(s,12,length(s)-11);
       ExtListBox.Items.loadfromfile(hs);
       listbox.Items.Delete(i);
       // an [i] die Strings aus ExtListBox einfügen
       for z:=ExtListBox.Items.Count-1 downto 0 do begin
        ListBox.Items.Insert(i,ExtListBox.Items[z]);
       end;
      end;
      inc(i);
     end;
     //Kommentare und Leerzeilen aus Liste entfernen
     i:=0;
     while i<= listbox.items.count-1 do begin
      s:=listbox.items[i];
      if (s='') or (s[1]=';') then begin
        listbox.items.delete(i);
        i:=i-1;
      end;
      inc(i);
     end; //while
     ListIndex:=0;
     CreateScreen;
end;
```

Zunächst wird die Listendatei in die ListBox „listbox“ geladen.
Nun wird die Liste nach Zeilen durchsucht, an deren Anfang das Schlüsselwort „LOADSCREEN“ steht.
Mit diesem Schlüsselwort ist es möglich bereits existierende Listen in eine andere einzufügen.
Wird so eine Zeile gefunden wird der Rest der Zeile (der Parameter) in die Variable hs kopiert.
In hs steht nun der Dateiname der externen Liste z.B. *c:\vfd-studio\lists\winamp.lst*
Die LOADSCREEN-Zeile kann nun aus listbox entfernt werden, während eine andere Listbox, nämlich ExtListBox die externe Liste lädt. Deren Zeilen werden nun an der Stelle der LOADSCREEN-Zeile eingefügt.
Schließlich wird die ListBox listbox noch einmal durchsucht um dabei alle Leer- und Kommentarzeilen zu entfernen (Kommentarzeilen beginnen mit einem Semikolon).

Die Liste ist nun vorbereitet. Jetzt wird die globale Variable ListIndex auf Null gesetzt. Diese Variable gibt die aktuell zu interpretierende Zeile an.
Und schließlich wird die Prozedur CreateScreen aufgerufen, die den ersten Screen erstellen wird.

CreateScreen wird an dieser Stelle jedoch noch nicht beschrieben, da wir später noch darauf zurückkommen und uns jetzt zunächst zwei Timerobjekte näher ansehen.

Der InfoTimer hat ein Intervall von 1 Sekunde und ist dafür verantwortlich die aktuelle CPU- und Speichernutzung zu ermitteln und sie den lokalen Variablen cu und mu (steht für **C**PU-Usage und **M**emory-Usage) zuzuweisen.
Weiterhin ruft er jedes Mal die Prozedur RefreshInfos auf, wodurch Informationen auf dem Display, die sich ständig ändern (z.B. Uhrzeit), aktualisiert werden. Dazu später mehr.
Und zudem sorgt er dafür, dass das Abspielen von Animationen unterbrochen wird, wenn die CUP-Last über die „IdleLevel-Grenze“ steigt und dies gewünscht ist (s. Checkbox „OnlyOnIdleBox“).

```
procedure TMainForm.InfoTimerTimer(Sender: TObject);
begin
     //für CPUMONITOR:
     collectcpudata;
     cu:=abs(GetCPUUsage(0)*100);
     if cu>100 then cu:=100; //man weiß ja nie...
     releaseCPUdata;
     //für RAMMONITOR:
     mu:=100-( (sysinfo.GetFreeMemory / 1048576+sysinfo.GetFreeVirtualMemory /
               1048576)/(sysinfo.GetTotalMemory / 1048576+sysinfo.GetTotalVirtualMemory / 1048576)*100);
     RefreshInfos;
     //Dieser Timer sorgt außderdem dafür, daß Animationen gestoppt werden, sobald
     //die durchschnittl. CPU-Last über die IdleLevel-Grenze steigt...
     if ScreenAnimated then begin
      if PlayOnlyOnIdle and (not isIdle) and (not AnimationPaused) then begin
       vfd.PauseAnimation;
       AnimationPaused:=true;
      end;
      //... und lässt die Animation wieder weiterspielen,
      //wenn die CPU-Last wieder unter das IdleLevel fällt.
      if ((not PlayOnlyOnIdle) or isIdle) and AnimationPaused then begin
       vfd.PauseAnimation;
       AnimationPaused:=false;
      end;
     end;  //if ScreenAnimated then begin
end;
```

Der UsageTimer hat ein Intervall von 2 Sekunden und arbeitet ebenfalls während der ganzen Programmlaufzeit.
Seine Aufgabe besteht darin die Arrays CPUGraph und RAMGraph immer wieder zu aktualisieren und die Monitor-Extrascreens bei Anfrage zu aktivieren.

Die Arrays speichern jeweils den ältesten Eintrag am Anfang (Index 0) und den jüngsten am Ende.
Um einen neuen Eintrag hinzuzufügen muss das ganze Array also quasi „links rotiert“ werden. Dies geschieht in For-Schleifen. Anschließend wird der aktuelle Wert in das letzte Arrayelement kopiert.
Der Timer ruft die Prozedur ShowExtraScreens auf, wenn eine der aufgelisteten Boolean-Variablen auf true gesetzt wurde. Auch dazu später noch mehr.

```
procedure TMainForm.UsageTimerTimer(Sender: TObject);
var
   i: integer;
begin
     //Graphen aktualisieren
     //RamGraph
     for i:=0 to 251 do
      RAMGraph[i]:=RAMGraph[i+1];
     RAMGraph[252]:=round(mu);
     //CPUGraph - Durchschnittliche CPU-Last
     for i:=0 to 448 do
      CPUusage[i]:=CPUusage[i+1];
     CPUusage[449]:=cu;
     //prüfe ob vorgefertigte Screen gewählt wurden
     if CPUMONITORenabled or CPUMONITORcharenabled
     or RAMMONITORenabled or RAMMONITORcharenabled
     then ShowExtraScreens;
end;
```

2. Anwender klickt auf das Symbol und öffnet Menü. Anwender wählt Menüpunkt „Hauptfenster“

Die Reaktion des SysTray-Icon auf eine Rechtsklick ist in der Prozedur SysTrayIconMsgHandler festgelegt:

```
procedure TMainForm.SysTrayIconMsgHandler(var Msg: TMessage);
var
   cp: TPoint;
begin
  // Ermitteln der aktuellen Cursorkoordinaten für Popupmenü
  getcursorpos(cp);
  PosX:=cp.x;
  PosY:=cp.y;
  try
  case Msg.lParam of
    WM_MOUSEMOVE:;
    WM_LBUTTONDOWN:;
    WM_LBUTTONUP:;
    WM_LBUTTONDBLCLK:;
    WM_RBUTTONDOWN:;
    WM_RBUTTONUP: mainform.PopupMenu.popup(PosX,PosY);
    WM_RBUTTONDBLCLK:;
    NIN_BALLOONSHOW:
    {Message wenn Sprechblase gezeigt wird}
      ShowMessage('NIN_BALLOONSHOW');
    NIN_BALLOONHIDE:
    {Message wenn die Sprechblase verschwindet (z.B. wenn das Icon entfernt wird).}
      ShowMessage('NIN_BALLOONHIDE');
    NIN_BALLOONTIMEOUT:
      ShowMessage('NIN_BALLOONTIMEOUT');
    NIN_BALLOONUSERCLICK:
      ShowMessage('NIN_BALLOONUSERCLICK');
  end
  except
  end;
end;
```

Sie verweist auf die Prozedur „popup“ des Popup-Menüs und übergibt dieser die Koordinaten des Cursors, wodurch das Popup-Menü an der Position des Cursors eingeblendet wird.

Beim Klick auf den Menüpunkt „Hauptfenster“ wird Show1Click aufgerufen:

```
procedure TMainForm.Show1Click(Sender: TObject);
begin
     windowstate:=wsnormal;
     mainform.show;
end;
```

Überflüssig zu sagen, dass diese Prozedur das Hauptfenster wieder sichtbar macht (vgl. HideTimer).

3. Anwender wechselt zur erweiterten Ansicht

Das Hauptfenster kann in zwei Modi angezeigt werden. Die normale Ansicht ist Standard und zeigt alle nötigen Bedienelemente. Die erweiterte Ansicht zeigt zudem die ListBox, in welcher die Anweisungen des aktuell gezeigten Screens blau markiert sind, eine Checkbox zur Einstellung ob Animationen bei hoher CPU-Last unterbrochen werden sollen und unten rechts die Dauer des aktuellen Screens in Sekunden. Klicken auf das kleine Panel unten rechts wechselt zwischen beiden Ansichten:

```
procedure TMainForm.ViewListPanelClick(Sender: TObject);
var
   LI: Integer;
begin
     if ViewListPanel.caption='<' then begin
      MainForm.Width:=260;
      ViewListPanel.caption:='>';
     end else begin
      MainForm.Width:=630;
      ViewListPanel.caption:='<';
     end;
     Mainform.Position:=poScreenCenter;
     // momentan ausgeführten Block markieren
     ListBox.MultiSelect:=false;
     ListBox.MultiSelect:=true;
     LI:=ListIndex;
     listbox.selected[LI]:=true;
     while (ListBox.Items[LI]<>'NEWSCREEN') and (LI>0) do begin
      dec(LI);
      listbox.selected[LI]:=true;
     end;
end;
```

Hinweis: normalerweise ist der WaitTimer für das Markieren des aktuellen Screens in der ListBox zuständig.

4. Es wird eine andere Liste geladen

Klicken auf Schaltfläche „Liste laden...“ ruft die OnClick-Prozedur dieses Buttons auf:

```
procedure TMainForm.ListTestButtonClick(Sender: TObject);
var
   LI: integer;
begin
     opendialog.InitialDir:=extractFileDir(application.exename)+'\Lists';
     if opendialog.Execute then begin
      WaitTimer.enabled:=false;
      StopButton.caption:='Stop';
      PopupStopButton.caption:='Stop';
      ListIndex:=0;
      LoadList(opendialog.filename);
      // momentan ausgeführten Block (hier der erste) markieren
      ListBox.MultiSelect:=false;
      ListBox.MultiSelect:=true;
      LI:=ListIndex;
      listbox.selected[LI]:=true;
      while (ListBox.Items[LI]<>'NEWSCREEN') and (LI>0) do begin
       dec(LI);
       listbox.selected[LI]:=true;
      end;
      WaitTimer.enabled:=true;
      ToDoList:=extractfilename(opendialog.filename);
     end;
end;
```

Diese Prozedur öffnet ein Öffnen-Dialogfenster zur Auswahl einer Listendatei (*.lst).
Wurde das Dialogfenster nicht abgebrochen, sondern tatsächlich eine Liste ausgewählt (Rückgabewert true) wird der WaitTimer, verantwortlich für das Anzeigen der Screens vorübergehend deaktiviert, während die Listendatei in die ListBox geladen wird. Dies übernimmtdie bereits beschriebene Prozedur LoadList.
Anschließend wird der aktuelle Screen-Anweisungsblock in der Listbox wieder markiert. Nach dem Laden einer neuen Liste ist dies natürlich immer der erste Block.

Danach kann der WaitTimer wieder aktiviert werden.
War zuvor die automatische Fortsetzung der Screens deaktiviert (d.h. nach Ablauf der ScreenTime des aktuellen Screens wird nicht der nächste angezeigt), so ist sie jetzt wieder eingeschaltet, weshalb auch die Beschriftungen des Stop-Buttons und des entsprechenden Menüpunktes gesetzt werden müssen.

Nun ist es an der Zeit auf die Prozedur CreateScreen zurückzukommen, die wie beschrieben ja am Ende der LoadList-Prozedur aufgerufen wird:

```
procedure TMainForm.CreateScreen;
begin
      ExtraTimer.Enabled:=false;
      CPUMONITORenabled:=false;
      CPUMONITORcharenabled:=false;
      RAMMONITORenabled:=false;
      RAMMONITORcharenabled:=false;
      while (listbox.items[ListIndex]<>'SCREENEND') do begin
       Interprete(listbox.items[ListIndex]);
       listbox.ItemIndex:=ListIndex;
       inc(ListIndex);
      end;
end;
```

Diese Prozedur hat die Aufgabe einen neuen Screen zu erstellen.
Zunächst werden alle eventuell aktivierten vorgefertigten Screens deaktiviert indem der ExtraTimer (zuständig für Anzeige von Festplatteninformationen) abgeschaltet wird und die Boolean-Variablen, zum Aktivieren eines CPU- oder RAM-Monitores auf false gesetzt werden (diese Screens werden wie bereits beschrieben durch den UsageTimer aktiviert).

Danach wird der Screenblock, beginnend von der aktuellen Zeile bis zur ersten „SCREENEND“-Zeile zeilenweise durchgegangen und jede Zeile mit der Prozedur Interprete „interpretiert“:

Interprete ist eine der wichtigsten Methoden und daher etwas umfangreicher – vom Ablauf her jedoch immer wieder gleich:

```
//Befehle - Stringinterpreter
procedure TMainForm.Interprete(s:String);
var
   p1,p2,p3,p4,p5,p6: string; //Parameter
   l: byte;  // Stringlänge
   GFX: TBitmap;
begin
     l:=length(s);

     if (s='NEWSCREEN') or (s='SCREENEND') then begin
      WaitTimer.enabled:=false;
      ExtraTimer.Enabled:=false;
      ClearInfoStrings;
      vfd.stopClock;           // Uhr anhalten
      vfd.StopAnimation;       // Animation anhalten
      ScreenAnimated:=false;   // Voreinstellung: Screen enthält keine Animation (kann von ANIMATION-
                                  Befehl geändert werden);
      vfd.ClearBitmap;         // Bitmapzeichenfläche löschen
     end
     else if Pos('SELECTSCREEN',s)= 1 then begin
      p1:=copy(s,14,1); // Screennummer ermitteln
      vfd.SelectScreen(strtoint(p1));
     end
     else if Pos('CLEARSCREEN',s)= 1 then begin
      p1:=copy(s,13,1); // Screennummer ermitteln
      vfd.ClearScreen(strtoint(p1));
      if p1='1' then vfd.ClearBitmap; //wenn GFX_Screen gelöscht wurde, auch Bitmapobjekt löschen
     end
     else if Pos('FADEOUT',s)= 1 then begin
      p1:=copy(s,9,l-8); // Geschwindigkeit ermitteln
      vfd.Fade(true, strtoint(p1));
     end
     else if Pos('FADEIN',s)= 1 then begin
      p1:=copy(s,8,l-7); // Geschwindigkeit ermitteln
      vfd.Fade(false, strtoint(p1));
     end
     else if Pos('LIGHT',s)= 1 then begin
      p1:=copy(s,7,1); // Helligkeitslevel ermitteln (einstellig)
      case strtoint(p1) of
       8: vfd.WriteCommand(CMD_Light_Lev8);
       7: vfd.WriteCommand(CMD_Light_Lev7);
       6: vfd.WriteCommand(CMD_Light_Lev6);
```

```
  5: vfd.WriteCommand(CMD_Light_Lev5);
  4: vfd.WriteCommand(CMD_Light_Lev4);
  3: vfd.WriteCommand(CMD_Light_Lev3);
  2: vfd.WriteCommand(CMD_Light_Lev2);
  1: vfd.WriteCommand(CMD_Light_Lev1);
 end;
end
else if Pos('CPUMONITOR',s)= 1 then begin
 CPUMONITORenabled:=true;
 showCPUMONITOR;
end
else if Pos('CPUMONCHAR',s)= 1 then begin
 CPUMONITORcharenabled:=true;
 showCPUMONITORchar;
end
else if Pos('RAMMONITOR',s)= 1 then begin
 RAMMONITORenabled:=true;
 showRAMMONITOR;
end
else if Pos('RAMMONCHAR',s)= 1 then begin
 RAMMONITORcharenabled:=true;
 showRAMMONITORchar;
end
else if Pos('PRINTERS',s)= 1 then begin
 showPrinterScreen;
end
else if Pos('DRIVES',s)= 1 then begin
 showDrivesScreen;
end
else if Pos('PAINTSTRING',s)= 1 then begin
 p3:=copy(s,l-1,2); // Y-Position
 p2:=copy(s,l-4,2); // X-Position
 p1:=copy(s,14,l-14-6);  // String zum Ausgeben
 if pos('$', p1)<>0 then begin   //Wenn der String ein Steuerzeichen '$' enthält
  InfoString(p1,strtoint(p2),strtoint(p3));
 end else vfd.PaintString(p1,strtoint(p2),strtoint(p3));  //kein Steuerzeichen
end
else if Pos('TEXTOUT',s)= 1 then begin
   p6:=copy(s,10,l-8);  //'TEXTOUT ' aus s entfernen
   p5:=p6;
 p1:=copy(p6,0, (Pos('''',p5))-1 );     // Text
 p2:=copy(s,12+length(p1),3);          // XDisplay  (dreistellig)
 p3:=copy(s,16+length(p1),2);          // YDisplay  (zweistellig)
 p4:=copy(s,19+length(p1),2);          // Schrifthöhe (zweistellig)
 p5:=copy(s,22+length(p1),l-28+length(p1)); //Schriftart
 if pos('$', p1)<>0 then begin   //Wenn der Text ein Steuerzeichen '$' enthält
  FontLabel.Font.Color:=clblack;
  FontLabel.Font.Name:=p5;
  FontLabel.Font.Size:=strtoint(p4);
  Fonted:=true;
  InfoString(p1,strtoint(p2),strtoint(p3));
 end else begin
  Fonted:=false;
  vfd.Bitmap.Canvas.Font.Color:=clBlack;
  vfd.Bitmap.Canvas.Font.Name:=p5;
  vfd.Bitmap.Canvas.Font.Size:=strtoint(p4);
  vfd.Bitmap.Canvas.TextOut(strtoint(p2),strtoint(p3),p1);
 end;
end
else if Pos('SCREENTIME',s)= 1 then begin
 p1:=copy(s,12,l-11);
 waittimer.Interval:=strtoint(p1)*100;
 SCR_Time_Label.caption:='ScreenTime: '+floattostr(WaitTimer.Interval/1000)+ ' Sek.';
 waittimer.Enabled:=true;
end
else if Pos('STRETCHBITMAP',s)= 1 then begin
 p1:=copy(s,15,l-14);           //Dateiname
 gfx:=TBitmap.Create;
 gfx.Width:=256;
 gfx.Height:=64;
 gfx.LoadFromFile(p1);
 vfd.bitmap.Canvas.StretchDraw(rect(0,0,256,64),GFX);
 gfx.free;
end
else if Pos('ANIMATE',s)= 1 then begin
  p3:=copy(s,l-3,4);  //Speed in ms ermitteln
  p2:=copy(s,l-7,3);  //X-Position
  p1:=copy(s,9,l-17); //Filename
  vfd.Animate(p1,strtoint(p2),strtoint(p3));
```

```
      if (not PlayOnlyOnIdle) or isIdle then begin  //nur wenn isIdle oder PlayOnlyOnIdle abgeschaltet –
                                                    andernfalls Befehl ignorieren
        AnimationPaused:=false;
       end else begin
        vfd.PauseAnimation;   //Animation anhalten
        AnimationPaused:=true;
       end;
       ScreenAnimated:=true;          // Screen enthält eine Animation (relevant für Timer)
      end
      else if Pos('PIXELBYTE',s)= 1 then begin
       p3:=copy(s,l,1);  //Letztes Zeichen= Y-Position (einstellig[0..7])
       p2:=copy(s,l-4,3);  //X-Position (dreistellig)
       p1:=copy(s,l-8,3); //Pixelbyte (dreistellig)
       vfd.SetPixelbyte(strtoint(p1),strtoint(p2),strtoint(p3));
      end
      else if Pos('TIME',s)= 1 then begin
       p2:=copy(s,l-3,4);  //Position
       p1:=copy(s,6,l-9); //Analog oder Digital?
       if p1= 'ANALOG' then vfd.Time(true, strtoint(p2))
       else vfd.time(false, strtoint(p2));
      end
      else if Pos('PAINTBITMAP',s)= 1 then begin
         p6:=copy(s,13,l-11);  //'PAINTBITMAP ' aus s entfernen
         p5:=p6;
       p1:=copy(p6,0, (Pos('?',p5))-1 );     // FileName    (? darf in Dateinamen nicht vorkommen)
       p2:=copy(s,15+length(p1),3);          // XDisplay
       p3:=copy(s,19+length(p1),1);          // YDisplay  (nur einstellig)
       p4:=copy(s,21+length(p1),3);          // Breite in Pixeln
       p5:=copy(s,25+length(p1),3);          // X-Position im Bild
       p6:=copy(s,29+length(p1),l-28+length(p1)); //MalFarbe
       BitmapToVFD(p1,strtoint(p2),strtoint(p3),strtoint(p4),strtoint(p5),stringtocolor(p6));
      end;
end;
```

Um diese komplexe Prozedur zu verstehen eignet sich wieder ein Beispiel:
Es sei angenommen der aktuelle Screenblock der Liste enthält folgende Zeilen:

```
NEWSCREEN
SELECTSCREEN 3
CLEARSCREEN 2
PAINTSTRING 'Tag: $DOW$' 03 01
ANIMATE c:\delphi\test\vfd-studio\bitmaps\30globus.bmp 128 0300
TEXTOUT 'CPU: $CPUSPEED$' 017 30 10 Lithograph
SCREENTIME 1200
SCREENEND
```

Dieser Screen zeigt in der rechten Displayhälfte eine Animation an, links oben den aktuellen Wochentag und darunter die Taktfrequenz der CPU.

Die erste Zeile NEWSCREEN würde von Interprete so bearbeitet werden:

```
WaitTimer.enabled:=false;
ExtraTimer.Enabled:=false;
ClearInfoStrings;
vfd.stopClock;           // Uhr anhalten
vfd.StopAnimation;       // Animation anhalten
ScreenAnimated:=false;   // Voreinstellung: Screen enthält keine Animation (kann von ANIMATION-Befehl
                            geändert werden);
vfd.ClearBitmap;         // Bitmapzeichenfläche löschen
```

Der ExtraTimer wird abgeschaltet (falls es ein vorgefertigert Screen angezeigt werden soll, würde er durch ein entsprechendes Schlüsselwort wieder aktiviert werden).
Der WaiTimer wird, falls er noch/wieder aktiviert sein sollte, ebenfalls deaktiviert, damit er während der Bearbeitung der folgenden Zeilen nicht „dazwischenfunkt“ – mehr dazu bei Zeile SCREENTIME.
Die InfoStrings werden gelöscht – wie bereits beschrieben enthält dieses Array Strings, mit Informationen, die ständig aktualisiert werden müssen.
Eine eventuelle Animation des vorherigen Screens wird beendet. Die Variable ScreenAnimated wird entsprechend auf false gesetzt.
Zudem wird das virtuelle Bitmapobjekt der vfd-Komponente gelöscht (vgl. TPortIOVFD-Dokumentation).

Die nächste Zeile SELECTSCREEN 3 hat bereits einen Parameter. Die lokalen Variable p1-p6 sind gedacht solche Parameter zwischenzuspeichern, bevor diverse Prozeduren oder Funktionen mit den Parametern aufgerufen werden. Die Parameter werden anhand von Positionen im String oder Leer- bzw. Sonderzeichen ermittelt – in diesem, Fall ist die Länge von "SELECTSCREEN " bekannt, weshalb der Parameter einfach das 14. Zeichen im String ist.

```
else if Pos('SELECTSCREEN',s)= 1 then begin
 p1:=copy(s,14,1); // Screennummer ermitteln
 vfd.SelectScreen(strtoint(p1));
end
```

SELECTSCREEN 3 würde also die vfd-Komponente veranlassen beide Display-Screens (Charakter- und Grafik-Screen) zu aktivieren.

CLEARSCREEN 2 hat ebenfalls einen Parameter zur Bestimmung des/der Screens und dient dazu den betreffenden Screen zu löschen:

```
else if Pos('CLEARSCREEN',s)= 1 then begin
 p1:=copy(s,13,1); // Screennummer ermitteln
 vfd.ClearScreen(strtoint(p1));
 if p1='1' then vfd.ClearBitmap; //wenn GFX_Screen gelöscht wurde, auch Bitmapobjekt löschen
end
```

Die Zeile PAINTSTRING 'Tag: DOW' 03 01 ist schon etwas komplexer, da sie drei Parameter hat:

```
else if Pos('PAINTSTRING',s)= 1 then begin
 p3:=copy(s,l-1,2); // Y-Position
 p2:=copy(s,l-4,2); // X-Position
 p1:=copy(s,14,l-14-6);  // String zum Ausgeben
 if pos('$', p1)<>0 then begin   //Wenn der String ein Steuerzeichen '$' enthält
  InfoString(p1,strtoint(p2),strtoint(p3));
 end else vfd.PaintString(p1,strtoint(p2),strtoint(p3));  //kein Steuerzeichen
end
```

Hier werden die Parameter von hinten ermittelt. Zur Erinnerung: 'l' ist die Länge des Strings!
Die Parameter haben dann folgende Werte:

p3 : „01“
p2 : „03“
p1: „Tag: DOW“

$-Zeichen werden benutzt um Schlüsselwörter zu markieren.
Hätte p1 kein $-Zeichen würde der Text nun mit PaintString direkt auf das Display ausgegeben werden, da er jedoch mehr als Null besitzt muss p1 mit InfoString auf Schlüsselwörter untersucht werden.
Daher folgend ein kurzer Ausflug zu dieser ebenfalls sehr wichtigen Prozedur:

```
//Infos, die sich nicht ständig ändern
procedure TMainForm.InfoString(s: string; x,y: byte);
var
   i: Integer; //Index im String
begin
     while pos('$PCNAME$',s)<>0 do begin
      i:= pos('$PCNAME$',s);
      delete(s,i,length('$PCNAME$'));
      insert(sysinfo.computername,s,i);
     end;
     while pos('$USERNAME$',s)<>0 do begin
      i:= pos('$USERNAME$',s);
      delete(s,i,length('$USERNAME$'));
      insert(sysinfo.GetCurrentUserName,s,i);
     end;
     while pos('$MEMORY$',s)<>0 do begin
      i:= pos('$MEMORY$',s);
      delete(s,i,length('$MEMORY$'));
      insert(inttostr(round(sysinfo.GetTotalMemory / 1048576)),s,i);
     end;
     while pos('$OS$',s)<>0 do begin
      i:= pos('$OS$',s);
      delete(s,i,length('$OS$'));
      insert(sysinfo.GetOperatingSystem,s,i);
     end;
     while pos('$CPUSPEED$',s)<>0 do begin
      i:= pos('$CPUSPEED$',s);
      delete(s,i,length('$CPUSPEED$'));
      if sysinfo.GetCPUSpeed<>'unbekannt' then insert(sysinfo.GetCPUSpeed,s,i)
      else insert(inttostr(round(sysinfo.CalcCPUSpeed)),s,i);
     end;
     while pos('$CPUVENDOR$',s)<>0 do begin
      i:= pos('$CPUVENDOR$',s);
      delete(s,i,length('$CPUVENDOR$'));
      insert(sysinfo.GetCPUVendorIdentifier,s,i);
     end;
     while pos('$CPUNAME$',s)<>0 do begin
      i:= pos('$CPUNAME$',s);
      delete(s,i,length('$CPUNAME$'));
      insert(sysinfo.GetCPUName,s,i);
     end;
     while pos('$CPUIDENT$',s)<>0 do begin
      i:= pos('$CPUIDENT$',s);
      delete(s,i,length('$CPUIDENT$'));
      insert(sysinfo.GetCPUIdentifier,s,i);
     end;
     while pos('$CPUCLASS$',s)<>0 do begin
      i:= pos('$CPUCLASS$',s);
      delete(s,i,length('$CPUCLASS$'));
      insert('x'+inttostr(ord(cpu.CPUInfo.cpuClass))+'86',s,i);
     end;
     while pos('$CPUMODEL$',s)<>0 do begin
      i:= pos('$CPUMODEL$',s);
      delete(s,i,length('$CPUMODEL$'));
      insert(cpu.cpuinfo.cpuModel,s,i);
     end;
     while pos('$CPUFAMILY$',s)<>0 do begin
      i:= pos('$CPUFAMILY$',s);
      delete(s,i,length('$CPUFAMILY$'));
      insert(cpu.cpuinfo.cpuFamily,s,i);
     end;
     while pos('$TIMEZONE$',s)<>0 do begin
      i:= pos('$TIMEZONE$',s);
      delete(s,i,length('$TIMEZONE$'));
      insert(sysinfo.GetTimeZone,s,i);
     end;
     while pos('$RECENTDD$',s)<>0 do begin
      i:= pos('$RECENTDD$',s);
      delete(s,i,length('$RECENTDD$'));
      insert(sysinfo.GetMostRecentDirectDraw,s,i);
     end;
     while pos('$RECENTD3D$',s)<>0 do begin
      i:= pos('$RECENTD3D$',s);
      delete(s,i,length('$RECENTD3D$'));
      insert(sysinfo.GetMostRecentDirect3D,s,i);
     end;
     //ProzessorFähigkeiten
        while pos('$MMX$',s)<>0 do begin
         i:= pos('$MMX$',s);
```

```
          delete(s,i,length('$MMX$'));
          if copy(cpu.FeatureList[24],10,3)='YES' then insert('ja',s,i)
          else insert('nein',s,i);
         end;
         while pos('$FPU$',s)<>0 do begin
          i:= pos('$FPU$',s);
          delete(s,i,length('$FPU$'));
          if copy(cpu.FeatureList[0],9,3)='YES' then insert('ja',s,i)
          else insert('nein',s,i);
         end;
         while pos('$VME$',s)<>0 do begin
          i:= pos('$VME$',s);
          delete(s,i,length('$VME$'));
          if copy(cpu.FeatureList[1],9,3)='YES' then insert('ja',s,i)
          else insert('nein',s,i);
         end;
         while pos('$PSN$',s)<>0 do begin
          i:= pos('$PSN$',s);
          delete(s,i,length('$PSN$'));
          if copy(cpu.FeatureList[18],9,3)='YES' then insert('ja',s,i)
          else insert('nein',s,i);
         end;
         while pos('$ISSE2$',s)<>0 do begin
          i:= pos('$ISSE2$',s);
          delete(s,i,length('$ISSE2$'));
          if copy(cpu.FeatureList[26],11,3)='YES' then insert('ja',s,i)
          else insert('nein',s,i);
         end;
         while pos('$ACC$',s)<>0 do begin
          i:= pos('$ACC$',s);
          delete(s,i,length('$ACC$'));
          if copy(cpu.FeatureList[29],9,3)='YES' then insert('ja',s,i)
          else insert('nein',s,i);
         end;
         while pos('$MSR$',s)<>0 do begin
          i:= pos('$MSR$',s);
          delete(s,i,length('$MSR$'));
          if copy(cpu.FeatureList[5],9,3)='YES' then insert('ja',s,i)
          else insert('nein',s,i);
         end;
         while pos('$PAE$',s)<>0 do begin
          i:= pos('$PAE$',s);
          delete(s,i,length('$PAE$'));
          if copy(cpu.FeatureList[6],9,3)='YES' then insert('ja',s,i)
          else insert('nein',s,i);
         end;
         while pos('$APIC$',s)<>0 do begin
          i:= pos('$APIC$',s);
          delete(s,i,length('$APIC$'));
          if copy(cpu.FeatureList[9],10,3)='YES' then insert('ja',s,i)
          else insert('nein',s,i);
         end;
       if (pos('$',s)<>0) then begin   // Wenn jetzt immer noch ein $ im String ist....
         if y<= 7 then begin
          Infostrings[y].Info:=s;
          InfoStrings[y].x:=x;
          InfoStrings[y].y:=y;
          RefreshInfos;
         end
         else begin
          vfd.Bitmap.Canvas.Font.Color:=clBlack;
          vfd.Bitmap.Canvas.Font.Name:=FontLabel.Font.Name;
          vfd.Bitmap.Canvas.Font.Size:=FontLabel.Font.Size;
          vfd.Bitmap.Canvas.TextOut(x,y,s);
          Fonted:=false;
         end;
       end
       else begin                    // ....anderfalls
        if Fonted=false then vfd.PaintString(s,x,y)
        else begin
         vfd.Bitmap.Canvas.Font.Color:=clBlack;
         vfd.Bitmap.Canvas.Font.Name:=FontLabel.Font.Name;
         vfd.Bitmap.Canvas.Font.Size:=FontLabel.Font.Size;
         vfd.Bitmap.Canvas.TextOut(x,y,s);
         Fonted:=false;
        end;
       end;
end;
```

InfoStrings ersetzt Schlüsselwörter durch Informationen, die sich nicht ständig ändern.
Dazu gehören z.B. Prozessorfeatures oder der Name des Betriebssystemes.

In dem in unserem Beispiel übergebenen String „Tag: DOW“ ist kein Schlüsselwort aus InfoStrings, aber immer noch mindestens ein $-Zeichen.
Folglich muss es sich um ständig zu aktualisierende Informationen handeln und es wird ein Eintrag im InfoStrings-Array erzeugt:

```
        Infostrings[y].Info:=s;
        InfoStrings[y].x:=x;
        InfoStrings[y].y:=y;
        RefreshInfos;
```

Die Position im Array (d.h. das Array-Element in das geschrieben wird) entspricht der Zeilennummer.
Danach wird RefreshInfos aufgerufen:

```
//Infos, die sich ständig ändern
procedure TMainForm.RefreshInfos;
var
   z: Integer;
   i: Integer;
   s: String;
   sec: String;  //Sekunden (Winamp)
begin
     for z:=0 to 7 do begin
      s:=Infostrings[z].Info;
      if s<>'' then begin    // nur wenn auch was im string steht
        while pos('$CPUUSAGE$',s)<>0 do begin
         i:= pos('$CPUUSAGE$',s);
         delete(s,i,length('$CPUUSAGE$'));
         insert(floattostr( round(abs( cu*10) )/10),s,i);
        end;
        while pos('$FREEMEM$',s)<>0 do begin
         i:= pos('$FREEMEM$',s);
         delete(s,i,length('$FREEMEM$'));
         insert(inttostr(round(sysinfo.GetFreeMemory / 1048576)),s,i);
        end;
        while pos('$AVERAGECPU$',s)<>0 do begin
         i:= pos('$AVERAGECPU$',s);
         delete(s,i,length('$AVERAGECPU$'));
         insert(format('%f',[averageCPUusage]),s,i);
        end;
        while pos('$IP$',s)<>0 do begin
         i:= pos('$IP$',s);
         delete(s,i,length('$IP$'));
         insert(sysinfo.GetIPAdress,s,i);
        end;
        while pos('$DOW$',s)<>0 do begin
         i:= pos('$DOW$',s);
         delete(s,i,length('$DOW$'));
         insert(sysinfo.GetDayOfTheWeek,s,i);
        end;
        while pos('$UPTIME$',s)<>0 do begin
         i:= pos('$UPTIME$',s);
         delete(s,i,length('$UPTIME$'));
         insert(sysinfo.GetUptime,s ,i);
        end;
        while pos('$RES$',s)<>0 do begin
         i:= pos('$RES$',s);
         delete(s,i,length('$RES$'));
         insert(sysinfo.GetDesktopResolution,s,i);
        end;
        while pos('$COLORS$',s)<>0 do begin
         i:= pos('$COLORS$',s);
         delete(s,i,length('$COLORS$'));
         insert(sysinfo.GetDesktopColors,s,i);
        end;
        while pos('$FREQ$',s)<>0 do begin
         i:= pos('$FREQ$',s);
         delete(s,i,length('$FREQ$'));
         insert(sysinfo.GetScreenFrequency,s,i);
        end;
        while pos('$WAVERSION$',s)<>0 do begin
```

```
          i:= pos('$WAVERSION$',s);
          delete(s,i,length('$WAVERSION$'));
          if winamp.IsWinampRunning then insert(Winamp.GetVersion,s,i)
          else insert('---',s,i);
         end;
         while pos('$WATITLE$',s)<>0 do begin
          i:= pos('$WATITLE$',s);
          delete(s,i,length('$WATITLE$'));
          if winamp.IsWinampRunning then insert(Winamp.GetTrackTitle,s,i)
          else insert('---',s,i);
         end;
         while pos('$WALENGTH$',s)<>0 do begin
          i:= pos('$WALENGTH$',s);
          delete(s,i,length('$WALENGTH$'));
          if Winamp.IsWinampRunning then begin
           sec:=inttostr(winamp.GetLength mod 60);
           if length(sec)<2 then sec:='0'+sec;
           insert(inttostr(winamp.GetLength div 60)+':'+sec,s,i)
          end else insert('0:00',s,i);
         end;
         while pos('$WAPOS$',s)<>0 do begin
          i:= pos('$WAPOS$',s);
          delete(s,i,length('$WAPOS$'));
          if winamp.IsWinampRunning then begin
           sec:=inttostr( (winamp.GetOffset div 1000000)mod 60);
           if length(sec)<2 then sec:='0'+sec;
           insert(inttostr( (winamp.GetOffset div 1000000)div 60)+':'+sec,s,i)
          end else insert('0:00',s,i);
         end;
         while pos('$DATE$',s)<>0 do begin
          i:= pos('$DATE$',s);
          delete(s,i,length('$DATE$'));
          insert(datetostr(now),s,i);
         end;
         while pos('$TIME$',s)<>0 do begin
          i:= pos('$TIME$',s);
          delete(s,i,length('$TIME$'));
          insert(timetostr(now),s,i);
         end;
         while length(s)<43 do s:=s+' ';   // String bis Displayende mit Leerstellen füllen
         vfd.PaintString(s,Infostrings[z].x,Infostrings[z].y);
       end; //if s<>''
      end;  //for z
end;
```

RefreshInfos ist zuständig dafür alle Einträge des InfoStrings-Array durchzugehen und in allen Einträgen Schlüsselwörter durch aktuelle Informationen zu ersetzen und diese schließlich via PaintStrings auf dem Display auszugeben.

Und hier wird auch endlich „DOW“ in unserem String gefunden und ersetzt. Aus „Tag: DOW“ wird so z.B. „Tag: Freitag“.

Damit nun aber genug davon und zurück zur Interprete-Prozedur!
Die nächste Zeile lautet: ANIMATE c:\delphi\test\vfd-studio\bitmaps\30globus.bmp 128 0300

Und folgende Anweisungen werden ausgeführt:

```
      else if Pos('ANIMATE',s)= 1 then begin
        p3:=copy(s,l-3,4);  //Speed in ms ermitteln
        p2:=copy(s,l-7,3);  //X-Position
        p1:=copy(s,9,l-17); //Filename
        vfd.Animate(p1,strtoint(p2),strtoint(p3));
      if (not PlayOnlyOnIdle) or isIdle then begin  //nur wenn isIdle oder PlayOnlyOnIdle abgeschaltet –
                                                    andernfalls Befehl ignorieren
        AnimationPaused:=false;
       end else begin
        vfd.PauseAnimation;   //Animation anhalten
        AnimationPaused:=true;
       end;
       ScreenAnimated:=true;          // Screen enthält eine Animation (relevant für Timer)
      end
```

Parameter p3 enthält die Zeit in Millisekunden die vergeht, bis das nächste Frame angezeigt wird, p2 gibt die x-Position der Animation auf dem Display an (Animationen müssen genau 64 Pixel hoch sein, weshalb eine y-Position entfällt) und p1 gibt den Dateinamen der Animationsbitmapdatei an.
Nicht vergessen sollte man dabei jedoch, dass es noch einen „versteckten“ Parameter im Dateinamen selbst gibt, der die Anzahl der Einzelbilder angibt (hier 30) – siehe Dokumentation TPortIOVFD.

Anschließend wird die Animate-Prozedur der vfd-Komponente mit den Parametern aufgerufen.
Die Animation wird allerdings sofort wieder unterbrochen, falls die CPU-Last hoch ist und Animationen nur im Idle-Mode abgespielt werden sollen. Im Extremfall wird die Animation somit erst gar nicht auf das Display gezeichnet!

Die folgende Screen-Zeile des Beispiels lautet: TEXTOUT 'CPU: $CPUSPEED$' 017 30 10 Lithograph

Interprete findet dazu diese Anweisungen:

```
else if Pos('TEXTOUT',s)= 1 then begin
 p6:=copy(s,10,l-8);  //'TEXTOUT ' aus s entfernen
 p5:=p6;
 p1:=copy(p6,0, (Pos('''',p5))-1 );     // Text
 p2:=copy(s,12+length(p1),3);          // XDisplay  (dreistellig)
 p3:=copy(s,16+length(p1),2);          // YDisplay  (zweistellig)
 p4:=copy(s,19+length(p1),2);          // Schrifthöhe (zweistellig)
 p5:=copy(s,22+length(p1),l-28+length(p1)); //Schriftart
 if pos('$', p1)<>0 then begin   //Wenn der Text ein Steuerzeichen '$' enthält
  FontLabel.Font.Color:=clblack;
  FontLabel.Font.Name:=p5;
  FontLabel.Font.Size:=strtoint(p4);
  Fonted:=true;
  InfoString(p1,strtoint(p2),strtoint(p3));
 end else begin
  Fonted:=false;
  vfd.Bitmap.Canvas.Font.Color:=clBlack;
  vfd.Bitmap.Canvas.Font.Name:=p5;
  vfd.Bitmap.Canvas.Font.Size:=strtoint(p4);
  vfd.Bitmap.Canvas.TextOut(strtoint(p2),strtoint(p3),p1);
 end;
end
```

Dies ist ein schönes Beispiel für eine etwas aufwendigere Parameterermittlung.
Diese ist notwendig, da zwei Parameter eine unbekannte Länge haben: der auszugebende Text und der Name der Schriftart.
Hier die Vorgehensweise:

Zunächst wird der String ohne „TEXTOUT '“ in p6 kopiert.
In p6 steht dann „CPU: $CPUSPEED$' 017 30 10 Lithograph“
p6 wird dann nach p5 kopiert.
Dann wird nach p1 der Text aus p6 bis zur Stelle des ersten '-Zeichens in p5 kopiert, so dass in p1 steht: „CPU: $CPUSPEED$“
p1 enthält nun den auszugebenden Text. Die nachfolgenden Parameter sind durch die Länge dieses Textes und die bekannte Länge von „TEXTOUT '“ ermittelbar.
p2 erhält so den Wert „017“, p3: „30“ und p4: „10“.
Nach p5 wird dann der Rest des Stringes bis zu seinem Ende kopiert, so dass dort steht: „Lithograph“.

Mit diesen Parametern kann dann Text auf dem Bitmapobjekt der vfd-Komponete gezeichnet werden, was durch das OnChangeEvent dieser Bitmap letztlich zur Ausgabe auf dem Display führt (s. TPortIOVFD).

Die beiden letzten Zeilen des Beispiels
SCREENTIME 1200
SCREENEND
sind jetzt schnell erklärt.

Das Schlüsselwort „Screentime“ dient dazu die Intervallzeit des WaitTimers zu setzen.
Der Parameter gibt die Anzeigedauer des Screens an und ist in Zehntelsekunden angegeben („600“ entspricht 1 Minute).
WaitTimer sorgt dafür, dass nach Ablauf dieser Zeit der darauffolgende Screen angezeigt wird.
Der zuvor deaktivierte WaitTimer kann nun auch wieder eingeschaltet werden.

```
else if Pos('SCREENTIME',s)= 1 then begin
 p1:=copy(s,12,l-11);
 waittimer.Interval:=strtoint(p1)*100;
 SCR_Time_Label.caption:='ScreenTime: '+floattostr(WaitTimer.Interval/1000)+ ' Sek.';
 waittimer.Enabled:=true;
end
```

SCREENEND markiert das Ende des aktuellen Anweisungsblockes.
Dieses Schlüsselwort wird jedoch nie interpretiert, da CreateScreen vor dieser Stelle die Interpretierung abbricht:

```
while (listbox.items[ListIndex]<>'SCREENEND') do begin
 Interprete(listbox.items[ListIndex]);
 listbox.ItemIndex:=ListIndex;
 inc(ListIndex);
end;
```

(aus CreateScreen)

Der Screen ist somit abgeschlossen.
Nach Ablauf der ScreenTime wird das TimerEvent des WaitTimers ausgelöst und damit seine Timer Prozedur WaitTimerTimer:

```
procedure TMainForm.WaitTimerTimer(Sender: TObject);
var
   LI: Integer;
begin
     waittimer.Enabled:=false;
     // Bei ListEnde wieder zum ListAnfang springen
     if ListIndex < ListBox.items.count-1 then inc(ListIndex)
     else ListIndex:=0;
     CreateScreen;
     // momentan ausgeführten Block markieren
     ListBox.MultiSelect:=false;
     ListBox.MultiSelect:=true;
     LI:=ListIndex;
     listbox.selected[LI]:=true;
     while (ListBox.Items[LI]<>'NEWSCREEN') and (LI>0) do begin
      dec(LI);
      listbox.selected[LI]:=true;
     end;
end;
```

Wie zu sehen deaktiviert der Timer sich selbst nach jedem Intervall, so dass er eigentlich nicht noch zusätzlich vom NEWSCREEN-Schlüsselwort deaktiviert werden müsste.
Dort wird er aber trotzdem sicherheitshalber nochmals deaktiviert, falls der Anwender just im Augenblick zwischen TimerEvent und CreateScreen auf den Stop-Button klickt.
Das Problem ließe sich sicher eleganter lösen, wenn man in dieser Zeit den Button deaktivieren würde – dies könnte dann jedoch u.U. zu einem Flackern der Schaltfläche führen.
Jedenfalls hat WaitTimer die Aufgabe zum nächsten Screen zu wechseln.
War der Aktuelle der letzte Screen der Liste, so wird ListIndex wieder auf Null gesetzt um wieder beim ersten Screen zu beginnen. Wenn nicht, entfällt dies und CreateScreen kann aufgerufen werden.
Abschließend wird der Screenblock in der Listbox wieder markiert.

Damit zurück zum Anwendungsszenario:

5. Klick auf Button „Nächster Screen >>“

Fast identisch mit der Funktion der WaitTimerTimer-Prozedur ist die Prozedur NextButtonClick, die beim Klicken auf die „Nächster Screen >>“-Schaltfläche ausgelöst wird:

```
procedure TMainForm.NextButtonClick(Sender: TObject);
var
   LI: Integer;
begin
     waittimer.Enabled:=false;
     StopButton.caption:='Stop';
     PopupStopButton.caption:='Stop';
     // Bei ListEnde wieder zum ListAnfang springen
     if ListIndex < ListBox.items.count-1 then inc(ListIndex)
     else ListIndex:=0;
     CreateScreen;
     // momentan ausgeführten Block markieren
     ListBox.MultiSelect:=false;
     ListBox.MultiSelect:=true;
     LI:=ListIndex;
     listbox.selected[LI]:=true;
     while (ListBox.Items[LI]<>'NEWSCREEN') and (LI>0) do begin
      dec(LI);
      listbox.selected[LI]:=true;
     end;
end;
```

Einziger Unterschied ist, daß der Buttons und der entsprechende Menüpunkt mit „Stop“ beschriftet wird.

6. Klick auf Button „Stop“ und
7. Klick auf Button „Go“

Der Stop-Button ist eigentlich viel mehr ein Pause-Button. Wird er angeklickt ändert er seine Beschriftung zwischen „Stop“ und „Go“.
Während er mit „Stop“ beschriftet ist, ist der WaitTimer aktiviert und wird nach Ablauf der ScreenTime zum nächsten Screen wechseln. Wünscht der Anwender den aktuellen Screen längere Zeit anzusehen, braucht er nicht die Listendatei, bzw. die ScreenTime zu editieren, sondern klickt einfach auf den Stop-Button (oder den entsprechenden Menüpunkt des Popupmenüs).
Der ändert seine Beschriftung dann zu „Go“ und hält den WaitTimer an, indem er ihn deaktiviert.
Um die automatische Screenfortführung wieder zu aktivieren genügt ein erneutes Klicken auf den Button:

```
procedure TMainForm.StopButtonClick(Sender: TObject);
begin
     WaitTimer.Enabled:=not WaitTimer.enabled;
     if Waittimer.enabled=true then begin
      StopButton.caption:='Stop';
      Popupstopbutton.caption:='Stop';
     end
     else begin
      StopButton.caption:='Go';
      PopupStopButton.caption:='Go';
     end;
end;
```

8. Klick auf Button „OK“

Durch Klicken auf den OK-Button verschwindet das Hauptfenster wieder in der Taskbar – es wird kurzerhand unsichtbar gemacht:

```
procedure TMainForm.OKButtonClick(Sender: TObject);
begin
     hide;
end;
```

9. Anwender klickt auf das Symbol und öffnet Menü. Anwender wählt Menüpunkt „Exit“

Und schließlich die Programmbeendung:
Durch Klick auf den Exit-Button oder den Menüpunkt „Exit“ wird das Hauptfenster MainForm geschlossen:

```
procedure TMainForm.Exit1Click(Sender: TObject);
begin
     mainform.close;
end;
```

Das bewirkt jedoch zudem den Aufruf der FormCloseQuery-Prozedur, die auch direkt aufgerufen wird, wenn z.B. der Rechner heruntergefahren wird.
Diese Prozedur schaltet beide Display-Screens ab, speichert die Programmeinstellungen in der *vfd.ini*-Datei, schließt den LPT-Treiber und entfernt das VFD-Studio-Symbol aus der Taskleiste bevor das Programm beendet wird:

```
procedure TMainForm.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
var
  VFDIni: TIniFile;
  IniName: string;
begin
    vfd.SelectScreen(0);     //VFD abschalten bei Programmende  (Stromversorgung und Heizdrähte sind
                               natürlich immer noch an!)
    IniName:=extractfilepath(application.ExeName)+'Vfd.ini';
    VFDIni := TIniFile.Create(IniName);
    with VFDIni do begin
       WriteInteger('SETTINGS', 'Port', mainform.LPTPort);
       WriteString('LIST', 'Listname',mainform.ToDoList);
       WriteBool('ANIMATIONS', 'PlayOnlyOnIdle', PlayOnlyOnIdle);
       WriteInteger('ANIMATIONS', 'IdleLevel', IdleLevel);
       WriteInteger('ANIMATIONS', 'IdleTime', IdleTime);
    end;
    VFDIni.Free;
    try
    dlportio.CloseDriver;
    except
    end;
    DeleteSysTrayIcon;
    application.Terminate;
end;
```

Das Anwendungsszenario ist somit beendet, und durch die Beispiele hoffentlich nachvollziehbar dargestellt.

Vorgefertigte Screens

Nun noch eine Kurzbeschreibung der vorgefertigten Screens.
Für die Monitor-Screens, welche den Verlauf von CPU- oder Speichernutzung darstellen, wird zunächst ein Rahmen auf das Display gezeichnet. Das Last-Diagramm wird dann von der Prozedur ShowExtraScreens gezeichnet. Diese Prozedur wird vom UsageTimer in Intervallen von zwei Sekunden aufgerufen, wenn einer der Monitor-Screens angezeigt werden soll.
Laufwerksinformationen werden vom ExtraTimer gezeigt. Die Anzeigedauer für ein einzelnes Laufwerk und die der Übersicht über alle Laufwerke kann in den Programmeinstellungen festgelegt werden.
Zur Visualisierung der freien Festplattenkapazität wird ein Bar-Graph von der Prozedur CharBar gezeichnet (Textmodus).
Die Druckerinformationen werden von der Prozedur showPrinterScreen direkt auf das Display gezeichnet.