OS2.x-Kurs

Björn Schotte

Copyright © 1994 by BOMBERSOFT

COLLABORATORS			
	TITLE : OS2.x-Kurs		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Björn Schotte	August 24, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

OS2.x-Kurs

1.1 OS2.x-Kurs by Björn Schotte

```
                ****************************************************************************

       #####   ####  #  # #####  ##### ##### #####  ####   ##### ######
       #    # #    # ##  ## #  # # #    # #      #    # #         ##
       #####  #    # # ## # #####  ### ##### ####  #    # ####     ##
       #    # #    # #  # #  # # #   #     # #    # #         ##
       #####   ####  #  # #####  ##### #    # #####   ####  #         ##


              presents:

          Der komplette OS2.x-Kurs auf AmigaGuide® !!

*****************************************************************************


Inhalt
******

1. Allgemeines


                 Adresse des Autors
                     Wo man den Autor erreichen kann

                 TagItems
                                Was sind TagItems ?

                 Danksagungen
                           Wem ich Danke sagen muß

                 Copyright
                              Wer trägt das Copyright ?

2. intuition.library
```

Screens und Windows
Wie öffne ich einen Screen ?

3. gadtools.library

Gadget-Konstruktion
Neue Erstellung von Gadgets !

OS2.x-Menüs
Konstruktion von fontsensitiven Menüs

1.2 Adresse des Autors

Wer mir eine Postkarte oder einen netten Brief schicken will (zwecks Bugreports, Fehlermeldungen, Laber-Letters, Geldsendungen etc.), der erreicht mich unter folgender Adresse:

Björn Schotte
Am Burkardstuhl 45
97267 Himmelstadt

1.3 tagitems

TagItems
****

Das sicherlich neue an OS2.0 sind die sog. TagItems. Diese sind wie folgt aufgebaut:

```
TagItem = RECORD
  ti_Tag  : LONG;
  ta_Data : LONG;
END;
```

Das Feld ti_Tag nimmt eine beliebige Konstante auf (z.B. WA_InnerWidth etc.), ti_Data dagegen die "Daten", die ti_Tag braucht. Diese Tags werden sehr oft vom System gebraucht !!

Siehe auch

CreateGadgetA-Tags
,
OpenScreenTagList-Tags
,

OpenWindowTagList-Tags

1.4 dankeschön

Wem ich alles DANKE sagen muß (und auch will !!):

'Bernd Künnen' (alias Diesel) für seine Puritys (Mach weiter so !!),

'Michael Klein' für seine tollen Proggys und Tips/Tricks,

'Rainer Behrens' (alias RabeSoft) für die Grüße in C.A.S.H. und der Shareware-Zahlung,

'Thomas Peifer' (alias PerfectSoft) für PBT V3.1M,

'Falk Zühlsdorf' (alias PackMan) für Cheque-Quellcode in one of the last Purities and ASL ...,

'Jan Stötzer' (alias Janosh/Dreamer) für seine diversen Beiträge auf den Puritys,

'Andreas Neumann' (alias Wurzelsepp) für Neptun, HAM etc.,

und alle anderen, die ich vergessen habe !!!

1.5 copyright

Dieser Kurs ist FREEWARE. Das Copyright liegt bei Björn Schotte. Allerdings bin ich gegen jede Spende etc. nicht abgeneigt. Ich hafte nicht für irgendwelche Schäden, die durch diesen Kurs auftreten könnten !! Aber ich bin für jede konstruktive Kritik empfänglich !!

Siehe

Adresse des Autors

1.6 Alles über Screens und Windows

Also, nun wollen wir mal einen Screen öffnen. Der DisplayModes soll HIRES sein und als Breite bzw. Höhe nehmen wir die Standardmaßen. Dann sieht ein Teil des Proggys so aus:

```
ta[1] := TagItem(SA_Width, STDSCREENWIDTH);
ta[2] := TagItem(SA_Height, STDSCREENHEIGHT);
ta[3] := TagItem(SA_DisplayID, HIRES_KEY);     { Auflösung }
```

mit:

```
ta : ARRAY[1..15] OF TagItem; { großzügig gewählt }
```

Nun wollen wir noch einen Titel für unseren Screen und die Tiefe setzen wir mal

```
auf 2 (4 Farben):

   { Definiert man es so, kann man bei bestimmten Tags in ti_Data einen String
     einsetzen !! }
   ta[4].ti_Tag := SA_Title;
   ta[4].ti_Data := "Erstes Beispiel-Programm zur OS2-Programmierung";
   ta[5] := TagItem(SA_Depth, 2);

So, nun wollen wir unseren Screen öffnen. Moment mal, wird hier jemand sagen..
Mußten wir nicht unter OS1.x die NewScreen-Struktur vollkommen ausfüllen, und
hier ??
Tja, der Vorteil von OS2.0 ist, daß ich nur die nötigsten Tags ausfüllen muß
bzw. kann, die anderen sind nämlich schon vordefiniert !!!
Also :

   { Die Tag-Liste müssen wir ja noch als "endlich" kennzeichnen: }
   ta[6].ti_Tag := TAG_DONE; { oder TAG_END }

   sp := OpenScreenTagList(NIL, ^ta);
   IF sp = NIL THEN { Fehler }

Der erste Parameter ist ein Zeiger auf eine NewScreen-Struktur, die dann mit
den Tags ergänzt wird. Das brauchen wir aber nicht. Nun haben wir unseren
Screen geöffnet. Es gibt aber noch viel mehr
                Screen-Tags
                .

Also, nun haben wir unseren Screen geöffnet, jetzt muß nur noch ein Fenster her:
Wie immer gibt es auch hier eine Unmenge von
                Window-Tags
                .

Also, öffnen wir ein Fenster:

  { Den IDCMP-Flags ist i.d.R. ein IDCMP_ vorangestellt !! }
  { Gleiches gilt für die Flags, d.h. WFLG_ vorangestellt !! }
  ta[1] := TagItem(WA_IDCMP, IDCMP_CLOSEWINDOW);
  ta[2] := TagItem(WA_Flags, WFLG_CLOSEGADGET+WFLG_ACTIVATE);
  ta[3] := TagItem(WA_CustomScreen, LONG(sp));
  ta[4].ti_Tag := WA_Title;
  ta[4].ti_Data := "Hallo";
  ta[5] := TagItem(WA_Top, 14);
  ta[6] := TagItem(WA_Height, sp^.Height - 14);
  ta[7].ti_Tag := TAG_DONE;

  wp := OpenWindowTagList(NIL,^ta);
  IF wp = NIL THEN { Fehler }

mit:
  wp : p_Window;
  sp : p_Screen;

Nun warten wir wie üblich mit WaitPort() und GetMsg() und replyen das ganze dann
mit ReplyMsg() und schliessen dann wie folgt:

  CloseWindow(wp);
  ok := CloseScreen(sp);
```

```
  { TRUE, falls Screen geschlossen werden konnte, andernfalls FALSE (Fenster noch
    auf dem Screen !!) }

mit:
  ok : BOOLEAN;
```

1.7 screentags

```
Hier die wichtigsten OpenScreenTagList-Tags. Angabe in den Klammern sind
die Default-Werte.

  TAG    BESCHREIBUNG
  ------------------------------------------------------------------------

  SA_Left  Position des Screens links (Default: 0)
  SA_Top   Position des Screens rechts (D: 0)
  SA_DetailPen   Textfarbe des Titels (D: -1)
  SA_BlockPen  Farbe des Titelbalkens (D: -1)
  SA_Font  Standard Zeichensatz des Screens (D: System Def.Font)
     { Zeiger auf eine TextAttr-Struktur erwartet:
           z.B.: :=TagItem(SA_Font, LONG(^tattr)); }
  SA_Type  CustomScreen oder WorkbenchScreen (D: CUSTOMSCREEN)
     { CUSTOMSCREEN oder WBENCHSCREEN }
  SA_BitMap  Zeiger auf eigene BitMap (D: NIL)
  SA_ShowTitle   Darstellung der Titelzeile bei Backdrop-Windows (D: LONG(FALSE) )
  SA_Behind  Screen beim Öffnen nicht nach vorne bringen (D: LONG(FALSE) )
  SA_Quiet   Der Screen wird ohne Titelbalken und Gadgets dargestellt
     (D: LONG(FALSE) )
  SA_Overscan  Einen der definierten Overscan-Größen wählen:
     OSCAN_TEXT, OSCAN_STANDARD, OSCAN_MAX oder OSCAN_VIDEO
     (D: OSCAN_TEXT)
  SA_DClip   Angezeigter Aussschnitt des Screens (DisplayClip-Region)
     { Zeiger auf Rectangle }
     (D: NIL)
  SA_Autoscroll  Sollte der Screen größer sein, als man auf
     dem Bildschirm darstellen kann, so wird mit
     diesem Boolean-Tag festgelegt, ob das Be-
     triebssystem den Bildschirm verschieben soll, wenn man mit dem
     Mauszeiger über den "Bildschirmrand" fährt.
     (D: LONG(FALSE) )
  SA_Pens  So, jetzt wird's spannend: Diesen Tag sollte man immer ver-
     wenden. Er legt nämlich fest, mit welcher Farbe was gezeichnet
     wird (Gadget-Rand hell/dunkel etc.). Man legt ein Array mit 9
     bzw. 10 Elementen des Typs INTEGER an und legt dann die Farben
     fest. Das letzte Element muß dann auf -1 zeigen. Die Standard-
     einstellung der Workbench ist dabei:

     3,1,1,2,1,3,1,0,2, -1

     DETAILPEN,BLOCKPEN,TEXTPEN,SHINEPEN,SHADOWPEN,FILLPEN,
     FILLTEXTPEN,BACKGROUNDPEN,HIGHLIGHTTEXTPEN

     Einfacher ist es, wenn man das erste Element auf -1 läßt,
```

```
    dann werden die Einstellungen der Workbench übernommen.

    (D: nix)

  SA_PubName   Name des Screens als Public-Screens (D: NIL)
  SA_PubTask   Dieser Task soll benachrichtigt werden, falls das letzte
    Fenster auf dem PublicScreen geschlossen worden ist. Nur
    gültig in Verbindung mit SA_PubSig (D: aufrufender Task)
  SA_PubSig  Signal, das per AllocSignal angefordert worden ist.
    (D: nix)
  SA_ErrorCode   Zeiger auf eine Variable, in der im "Fehlerfall" der
    Fehlercode abgelegt wird:

     OSERR_NOMONITOR      - angeforderter Monitor nicht vorhanden
     OSERR_NOCHIPS        - benötigte Customchips nicht vorhanden
     OSERR_NOCHIPMEM      - nicht genug CHIP-Memory
     OSERR_NOMEM        - kein Speicher frei
     OSERR_UNKNOWNMODE    - DisplayID unbekannt
     OSERR_PUBNOTUNIQUE   - Name des PublicScreens existiert
          schon

  ------------------------------------------------------------------------
```

1.8 windowtags

```
Ich habe hier einmal die wichtigsten OpenWindowTagList-Tags zusammen-
gestellt. Die Angabe in den Klammern (nach "D: ") stellt den Default-
Wert dar, der verwendet wird, falls das Tag nicht angegeben wird.

  TAG       ERKLÄRUNG
  ------------------------------------------------------------------------

  WA_Left     Linke Position (D: 0)
  WA_Top      Obere Position (D: 0)
  WA_Width      Breite (D: ScreenWidth)
  WA_InnerWidth     Breite des Fensters ohne (!) Rahmen (zu bevorzugen)
  WA_Height     Höhe (D: ScreenHeight)
  WA_InnerHeight    Höhe des Fensters ohne (!) Rahmen (zu bevorzugen)
  WA_DetailPen      Textfarbe d. Titels (D: -1)
  WA_BlockPen     Hintergrundfarbe d. Titels (D: -1)
  WA_IDCMP      IDCMP-Flags des Fensters (D: 0)
  WA_Flags      Flags des Fensters (D: 0)
  WA_Gadgets      Zeiger auf Gadget-Liste, die ans Fenster angehängt werden
       { p_Gadget } (D: NIL)
  WA_CheckMark      p_Image : Zeiger auf ein Image, das den Haken für Menüitems
       definiert (D: NIL)
  WA_Title      Unser Titel (D: nix)
  WA_CustomScreen   Übergeben wird hier unser Screen (nix)
  WA_SuperBitMap    Zeiger auf SuperBitMap { p_BitMap } (nix)
  WA_MinWidth     Minimale Breite (D: WA_Width)
  WA_MaxWidth     Maximale Breite (D: WA_Width)
  WA_MinHeight      Minimale Höhe (D: WA_Height)
  WA_MaxHeight      Maximale Höhe (D: WA_Height)
  WA_ScreenTitle    ScreenTitle bei aktivem Fenster
  WA_AutoAdjust     (LONG(TRUE) oder LONG(FALSE) ) -> Anpassung der Fenster-
```

```
        größe an den Screen erlaubt
  WA_PubScreenName  Name des PublicScreens, auf dem das Fenster
        erscheinen soll
  WA_PubScreen      Zeiger auf PublicScreen, auf dem das Fenster erscheinen
        soll (muß mit LockPubScreen gesichert werden !!)
  WA_MouseQueue     maximale Anzahl der IDCMP_MOUSEMOVE-Messages, die von
        Intuition aneinandergereiht werden (D: 5)
  ------------------------------------------------------------------------
```

1.9 (Fast) alles über die gadtools.library

Nun haben wir schon Screens und Fenster geöffnet, aber das alleine sieht noch nicht gut aus. Jetzt wollen wir Gadgets und Menüs in unserem Fenster installieren:

Also, als erstes brauchen wir die sog. VisualInfo. VisualInfo ?? Richtig gehört: Sie braucht man zum Screenspezifischen Zeichnen von Gadgets und Menüs. Und wie bekommen wir die ?? Ganz einfach, mit der Funktion GetVisualInfoA() aus der GadTools.library:

```
  { Am Anfang des Proggys: }
  OpenLib(gadtoolsbase,"gadtools.library",36);

  vi := GetVisualInfoA(ps, NIL);
```

mit:

```
  vi : PTR;
  ps : p_Screen;
```

Doch von welchem Screen brauchen wir die ? Vorausgesetzt, daß wir unser Fenster auf einem eigenen Screen öffnen, setzen wir unseren Screen-Pointer ein. Sollte unser Fenster auf der Workbench geöffnet werden, so müssen wir den Workbench-Screen-Zeiger einsetzen. Wie das geht ? Ganz einfach:

```
  ps := LockPubScreen("Workbench"); { Anstatt "Wor..." auch NIL }
  IF ps = NIL THEN { Fehler }
```

LockPubScreen ?? Tja, diese Funktion ist ähnlich der von LockIBase. Doch hier wird der Public (öffentliche) Screen gesperrt, so daß keine Fenster etc. geöffnet werden können bzw. der Screen geschlossen werden kann, um somit einige Daten zu bekommen. Wie bei LockIBase müssen wir, wenn wir unsere VisualInfo haben, den PubScreen wieder entriegeln. Somit sieht das eigentliche Fragment so aus:

```
  { Weiter oben Screen öffnen, falls gewünscht }

  ps := LockPubScreen(NIL);
  IF ps = NIL THEN { Fehler }
  vi := GetVisualInfoA(ps, NIL);
  UnLockPubScreen(NIL, ps);
  IF vi = NIL THEN { Fehler }
```

So, sollte alles gut laufen, haben wir jetzt unsere VisualInfo. Nun wollen

wir Gadgets in unser Fenster einbauen. Wie unter OS1.x möglich, kann man entscheiden, ob man direkt beim Öffnen die Gadgets in die Window-Gadget-List einhängt (WA_Gadgets !!) oder erst später per AddGList/AddGadget...
Ich ziehe den ersteren Fall vor, allerdings müssen wir dann hier erst die Gadgets definieren und dann (!) das Fenster öffnen.
Wir verwenden aber zum "Bauen" keine Gadget-Struktur, sondern definieren uns die NewGadget-Struktur, die man so definiert:

```
  ng := NewGadget(10,20,                        { LeftEdge, TopEdge }
      200,13,          { Breite, Höhe }
      "Hallo",^tattr,              { GadgetText, TextAttr }
      1,PLACETEXT_IN,        { GadgetNumber, Flags }
      vi,NIL);           { VisualInfo, UserData }
```

Einige Elemente sind sicher schon bekannt (Gadget-Struktur !!). Also, was wir brauchen, ist die VisualInfo (die holen wir uns vom PubScreen) und evtl. eine TextAttr-Struktur. Beispiel: Wir definieren uns ein Gadget, dann sieht unser Programmfragment so aus:

```
  gl := NIL;
  gl := CreateContext(^gl);   { MEGA-WICHTIG !! }
  IF gl=NIL THEN { Fehler }

  ng := NewGadget(10,20,
      100,13,
      "_Start",^tattr,    { "_" ?? siehe unten }
      1,PLACETEXT_IN,
      vi,NIL);

  { "_" ?? : }
  t[1] := TagItem(GT_Underscore, LONG("_"));
  { Aha, hier definieren wir das "Erkennungszeichen", welcher Buchstabe unter-
    strichen werden soll (in unserem Beispiel "S") }

  gad1 := CreateGadgetA(BUTTON_KIND,gad,^ng,^t);

  { CreateGadgetA(Art_des_Gadgets,Vorgänger_Gadget, }
  {     Zeiger_auf_Record,Evtl_Tags);     }

  IF gad1 = NIL THEN { Fehler }

  { Fenster öffnen; Tag WA_Gadgets:

    t[x] := TagItem(WA_Gadgets, LONG(gl));   }
```

Alles klar ??

Am besten ist, wenn ich Euch einmal die verschiedenen Gadgettypen vorstelle:

```
    BUTTON_KIND   : Ein ganz "normales" Gadget.
    STRING_KIND   : Ein String-Gadget.
    CHECKBOX_KIND : Ein "ToggleBool" - Gadget, daß nur zwei Zustände
        kennt: An oder Aus (Häkchen, kein Häkchen !!).
    CYCLE_KIND    : Ein Gadget, bei dem man zwischen verschiedenen
        Einstellungen durchklicken kann.
    INTEGER_KIND  : Wie STRING_KIND, nur daß man hier NUR Zahlen
```

```
        eingeben kann.
     LISTVIEW_KIND : Ermöglicht es, mehrere Einträge in einer Liste zu
         verwalten. Zusätzlich wird hier noch ein Scroll-
         Gadget zur Verfügung gestellt.
     MX_KIND     : Radiobuttons, kleine quadratische Schalter, die im
         aktivierten Zustand ausgefüllt sind.
     NUMBER_KIND   : Numerisches Gadget. Ist nur Read-Only. Kann als
         Informationsgadget dienen (z.B. Speicherplatz etc.)
     PALETTE_KIND  : Ein Palette-Gadget, mit dem man die einzelnen
         Farbregister auswählen kann (Tja, Michael, jetzt ist
         wohl ein neues RPalette fällig !!! :)
     SCROLLER_KIND : Erzeugt einen Scroll-Balken und dient der Bewegung
         von Listen und Bereichen.
     SLIDER_KIND   : -> PropGadget

Tja, das waren alle Gadgettypen !! Sicher gibt es noch
                Tags für die Gadgets
                .
Kreiert wird ein Gadget mit:

      gad := CreateGadgetA(what_a_kind,vorgänger,p_NewGadget,p_TagItem);

      z.B.: BUTTON_KIND,gl,^ng,^t);

    Haben wir also alle unsere Gadgets kreiert, werden diese entweder ans
    Fenster per AddGList angehängt oder per WA_Gadget-Tag gleich beim
    Öffnen des Fensters dargestellt.

    Wie modifiziere ich die Tags eines Gadgets nachträglich ?
    Kein Problem, nämlich mit der Prozedur GT_SetGadgetAttrsA(). Parameter:

    GT_SetGadgetAttrsA(gad:p_Gadget; win:p_Window; req:p_Requester; t:PTR);

    gad ist unser zu modifizierendes Gadget,
    win unser Window, auf dem das Gadget "liegt",
    req der Requester, in dem das Gadget dargestellt wird (Norm.: NIL) und
    ein Zeiger auf unsere TagItems, die neue Werte beinhalten !!

    Am Ende soll man natürlich aufräumen:

      CloseWindow(wp);
      FreeGadgets(gl);
      FreeVisualInfo(vi);

    Mit FreeGadgets() wird der Speicherplatz für die Gadgets wieder freigegeben.
```

1.10 Tags für z.B. CreateGadgetA()

```
Für alle Gadgets

  GT_Underscore      : Zeichen, das als Erkennungszeichen für den darauf-
           folgenden Buchstaben zum Unterstreichen dient.
  GA_Disabled      : TRUE, um das Gadget zu deaktivieren.
```

Checkbox-Gadget

```
     GTCB_Checked    : TRUE, damit das Gadget das Häkchen zeigt.
```

Cycle-Gadget

```
     GTCY_Active      : Nummer der aktiven Auswahl beginnend mit 0 (D: 0)
     GTCY_Labels      : Zeiger auf ein NIL-terminiertes STR-Feld, daß die
            Labels definiert.
```

Integer-Gadget

```
     GTIN_MaxChars      : Maximalanzahl von Ziffern.
     GTIN_Number      : Default-Wert des Gadgets (D: 0).
     STRINGA_Justification  : Ausrichtung der Zahl: GACT_STRINGLEFT,
            GACT_STRINGCENTER oder GACT_STRINGRIGHT
            (erst ab OS2.04 !!)
```

ListView-Gadget

```
     GTLV_Labels      : Zeiger auf eine verkettete Liste (p_List) von
            Node-Strukturen, deren Feld ln_name die Zeichen-
            kette enthält, die dargestellt wird.
     GTLV_ReadOnly      : TRUE, wenn kein Element der Liste ausgewählt
            werden kann (D: FALSE [0]).
     GTLV_ScrollWidth     : Breite des Scrollbar-Gadgets; muß >0 sein !!
     GTLV_Selected      : Nummer des ausgewählten Listenelements beginnend
            bei 0 (D: -1 [kein Element ausgewählt]).
     GTLV_ShowSelected      : Sollte auf 0 gestellt werden, damit man sieht,
            welches Element ausgewählt wurde.
     GTLV_Top       : Nummer des obersten sichtbaren Listenelements
            beginnend bei 0 (D: 0).
     LAYOUTA_SPACING      : Abstand zwischen den Zeilen (D: 0).
```

Radiobutton-Gadget

```
     GTMX_Active      : Nummer des aktiven RB-Elements beginnend mit 0
           (D: 0).
     GTMX_Labels      : Wie bei CycleGadgets (NIL-terminiertes STR-Feld).

     GTMX_Spacing     : Abstand zwischen den einzelnen Radionbuttons
            (D: 1).
```

ReadOnly-Integer-Gadget

```
     GTNM_Border      : TRUE, um einen vertieften Rahmen um das Gadget zu
            zeichnen.
     GTNM_Number      : Nummer, die dargestellt werden soll (D: 0).
```

Farbpaletten-Gadget

```
     GTPA_Color       : Ausgewähltes Farbregister (D: 1).
     GTPA_ColorOffset     : Erstes verwendetes Farbregister (D: 0).
     GTPA_Depth       : Anzahl der Bitplanes (D: 1).
     GTPA_IndicatorHeight   : Höhe der Farbanzeige.
     GTPA_IndicatorWidth    : Breite der Farbanzeige.


Scrollbar-Gadget

     GA_Immediate     : TRUE, falls IDCMP_GADGETDOWN-Messages gesendet
            werden sollen (D: 0 [=FALSE]).
     GA_Relverify     : TRUE, falls IDCMP_GADGETUP-Messages gesendet
            werden sollen (D: 0).
     GTSC_Arrows      : Breite der Pfeilgadgets (horizontales Scrollbar)
            bzw. Höhe (vertikales Scrollbar-Gadget).
     GTSC_Top       : Oberstes sichtbares Element (D: 0).
     GTSC_Total       : Gesamtzahl der Elemente (D: 0).
     GTSC_Visible     : Anz. der sichtbaren Elemente (D: 2).
     PGA_Freedom      : LORIENT_VERT für einen vertikalen oder
            LORIENT_HORIZ für einen horizontalen Scrollbar.


Slider-Gadget

     GA_Immediate     : s.o.
     GA_Relverify     : s.o.
     GTSL_Level       : Aktueller Wert des Sliders (D: 0).
     GTSL_LevelFormat     : Format-Zeichenkette (wie bei RawDoFmt!!) für
            die Ausgabe des aktuellen Wertes (D: keine)
     GTSL_LevelPlace      : Positionierung des Ausgabetextes:
            PLACETEXT_LEFT, _RIGHT, _ABOVE, _BELOW
           (D: PLACETEXT_LEFT).
     GTSL_Max,GTSL_MIN      : Maximal- und Minimalwert des Sliders (D: 15, 0).
     GTSL_MaxLevelLen     : Maximallänge des Textes, der den aktuellen
            Wert ausgibt (D: nix).
     PGA_Freedom      : siehe oben.


String-Gadget

     GTST_MaxChars      : Maximalanzahl von Zeichen ohne das abschliessende
            Nullbyte (D: 64).
     GTST_String      : Zeiger auf den Text des Gadgets (D: NIL).
     STRINGA_Justification  : GACT_STRINGLEFT, GACT_STRINGRIGHT,
            GACT_STRINGCENTER (erst ab OS2.04 !!).



ReadOnly-String-Gadgets

     GTTX_Border      : TRUE für vertieften Rahmen (D: FALSE).
     GTTX_CopyText      : TRUE, falls die initialisierende Zeichenkette
            in einen internen Puffer des Gadgets kopiert
            werden soll, anstelle nur den Zeiger zu verwenden;
            darf nicht angegeben werden, wenn für GTTX_Text
            NIL übergeben wird; dieses Tag wird erst ab OS
            2.04 unterstützt !!
```

```
     GTTX_Text        : Zeiger auf zu initialisierenden Text (D: NIL).
```

Nun, das waren die Tags (ein paar habe ich weggelassen, da sie für unsere Fälle uninteressant sind !!).

1.11 Erstellung von systemkonformen Menüs

Wie erstellt man Menüs ? Nun, dazu muß man sagen, daß es sehr sehr viel einfacher vonstatten geht wie unter OS1.3. Ein Menü wird ganz normal an ein Fenster angehängt mit SetMenuStrip(). Das Menü muß aber erst einmal definiert werden:

Mittels eines Arrays [] of
NewMenu
werden die Menüs definiert.

Kreiert wird dann das Menü mit CreateMenusA. Die Parameter sind:
Als erstes den Pointer auf unser NewMenu-Array. Als zweites sogenannte

CreateMenus-Tags
.
Die Funktion retourniert einen Zeiger auf ein Menu (p_Menu). Diesen Zeiger übergeben wir der Funktion LayoutMenusA, deren Parameter sind:
1. Der Zeiger, der von CreateMenusA zurückgeliefert wurde.
2. Die VisualInfo
3.
LayoutMenus-Tags
Retourniert die Funktion TRUE, so kann man mit SetMenuStrip das Menü ganz normal an ein Fenster hängen. LayoutMenusA wird insofern benötigt, als daß das Menü fontsensitiv layoutet wird !!

Das war's auch für das Konstruieren von Menüs !!

1.12 Der Record NewMenu

NewMenu sieht folgendermaßen aus:

```
  NewMenu = RECORD
    nm_Type      : Byte;
    nm_Pad       : Byte;
    nm_Label       : Str;
    nm_CommKey       : Str;
    nm_Flags       : Word;
    nm_MutualExclude : LongInt;
    nm_UserData      : Ptr;
  END;
```

```
  nm_Type:     NM_Title für einen Menütitel,
       NM_ITEM  für einen Menüpunkt,
```

```
       IM_ITEM  für einen Menüpunkt mit Image,
       NM_SUB   für einen Untermenüpunkt,
       IM_SUB   für einen Untermenüpunkt mit Image,
       NM_END   kennzeichnet das Ende der Liste und muß IMMER mit
          angegeben werden !!

  nm_Pad:    dient KickPascal zur internen Ausrichtung auf, ich glaub,
       Wortgrenze. Auf 0 setzen.

  nm_Label:    Text des Menüs, Items oder Subitems

  nm_CommKey:    ist die "Abkürzung" für den Punkt in Form von
       LAMIGA + <nm_CommKey>

  nm_Flags:    NM_MENUDISABLED, um ein Menü zu sperren,
       NM_ITEMDISABLED, um einen Menüpunkt zu sperren,
       CHECKIT, um einen Menüpunkt zu erzeugen, der bei Auswahl
       mittels eines CheckMark-Symbols gekennzeichnet wird,
       MENUTOGGLE, um einen Menüpunkt ein- und ausschaltbar zu
       machen,
       CHECKED, um einen Menüpunkt einzuschalten und damit das
       CheckMark-Symbol zu setzen.

  nm_MutualExclude:BitMaske, die einander ausschließende Menüelemente
       beschreibt.

  nm_UserData    Platz für eigene Funktion...
```

1.13 CreateMenus-Tags

```
Tags für Menüs und Menüpunkte

 GTMN_FrontPen
   Farbregister, das für den Menütext verwendet werden soll (Default: 0).
 GTMN_FullMenu
   TRUE, um festzulegen, daß die Angaben der NewMenu-Strukturen eine vollständige
   Menüleiste und nicht nur ein Fragment definiereren; ist dies dennoch der Fall,
   so liefert die Funktion einen Sekundärfehler mit dem Wert GTMENU_INVALID;
   dieses Tag wird erst ab OS-Version 2.04 unterstützt (Default: FALSE).
 GTMN_SecondaryError
   Zeiger auf eine Variable, die nach dem Aufruf der Funktion einen
   Sekundärfehlercode enthält; dieses Tag wird erst ab OS-Version 2.04 unterstützt
   (Default: NULL). Mögliche Fehlerwerte sind:
   GTMENU_INVALID, falls die übergebenen NewMenu-Strukturen ein ungültiges Menü
   definieren und der Funktionsaufruf deshalb erfolglos war;
   GTMENU_TRIMMED, falls das Menü zuviele Elemente besitzt und deshalb reduziert
       wurde;
   GTMENU_NOMEM bei Speicherplatzmangel.
```

1.14 LayoutMenus-Tags

```
Tags der Funktionen LayoutMenus() und LayoutMenusA()

Tags für Menüs
  GTMN_TextAttr
    Zeiger auf eine TextAttr-Struktur, die den Zeichensatz beschreibt, der für
    das Menü verwendet werden soll; dieser muß sich mittels OpenFont() öffnen
    lassen.
```

1.15 index

```
                INDEX
*****
```