```
gadgets.c -- execute me to compile me
=near nominc strmer streq nostkchk saveds ign=73 newgadgets
ROM LIB:c.o newgadgets.o TO newgadgets LIB LIB:sc.lib LIB:amiga.lib


de is only designed to demonstrate some of the new 3.0 GadTools
s. It does not handle all of the IDCMP events and other conditions like
ing pens for the palette gadget that a good user interface should.

so, that it takes advantage of the SAS 6.0 C compiler.



e <exec/types.h>
e <exec/memory.h>
e <exec/nodes.h>
e <intuition/intuition.h>
e <intuition/intuitionbase.h>
e <intuition/gadgetclass.h>
e <libraries/diskfont.h>
e <libraries/gadtools.h>

e <clib/intuition_protos.h>
e <clib/exec_protos.h>
e <clib/dos_protos.h>
e <clib/alib_protos.h>
e <clib/diskfont_protos.h>
e <clib/graphics_protos.h>
e <clib/gadtools_protos.h>

e <stdio.h>

LATTICE

void)

n (0);
                        /* Disable Lattice CTRL/C handling */


t (void)

n (0);
                        /* really */



are the library base pointers */
struct IntuitionBase *IntuitionBase;
struct Library *DiskfontBase, *GfxBase, *GadToolsBase;

 of gadgets used here */
Gadget *glist = NULL, *stringgad, *textgad, *checkgad, *mxgad, *palgad,
d,
d, *numgad, *slid1gad, *slid2gad, *slid3gad;

gn the gadget ids even though we're not using them */
MYGAD_STRING        69
MYGAD_TEXT          70
MYGAD_CHECK         71
MYGAD_MX            72
MYGAD_PAL           73
MYGAD_LIST          74
MYGAD_INT           75
MYGAD_NUM           76
MYGAD_SLIDERLEFT    77
MYGAD_SLIDERCENTER  78
MYGAD_SLIDERRIGHT   79

List *DaList;           /* the listview list */

olors[7] =
15, 6, 0, 3, 8};   /* the palette colortable */
```

```
/*
 *  Create the listview list
 */

BOOL
CreateDaList (UBYTE * names[])
{
  struct Node *DaListNode;
  USHORT i;
  BOOL okay = TRUE;

  if (DaList = AllocMem (sizeof (struct List), MEMF_FAST))
  {
    NewList (DaList);

    for (i = 0; names[i] != NULL && okay == TRUE; i++)
    {
      if (DaListNode = AllocMem (sizeof (struct Node), MEMF_FAST))
      {
        DaListNode->ln_Name = names[i];
        DaListNode->ln_Type = 100L;
        DaListNode->ln_Pri = 0;
        AddTail (DaList, DaListNode);
      }
      else
      {
        printf ("Could not allocate memory for node\n");
        okay = FALSE;
      }
    }
  }
  else
  {
    printf ("Could not allocate memory for list\n");
    okay = FALSE;
  }

  return (okay);
}


/*
 * Create the gadgets.
 *   The gadgets are more or less positioned relative to the window size
 *   and each other.  This is not extremely sophisticated, so don't expect
 *   them to look good on a lores screen.
 */

struct Gadget *
CreateGads (struct Screen *scr, struct VisualInfo *vi,
            struct TextAttr *gadfont)
{
  struct Gadget *gad;
  struct NewGadget ng;
  UBYTE *BassLines[] =
  {"Always", "Fine", "Do", "Boys", "Good", NULL};       /* mx choices */
  UBYTE *BassSpaces[] =
  {"Grass", "Eat", "Cows", "All", NULL};        /* listview choices */

  if (gad = CreateContext (&glist))
  {
    /* create an MX gadget */
    ng.ng_TextAttr = gadfont;
    ng.ng_VisualInfo = vi;
    ng.ng_LeftEdge = 80 + scr->WBorLeft;
    ng.ng_TopEdge = 30 + scr->WBorTop + (scr->Font->ta_YSize + 1);
    ng.ng_Width = 50;
    ng.ng_Height = 30;
    ng.ng_GadgetText = "Bass Lines";
    ng.ng_GadgetID = MYGAD_MX;
    ng.ng_Flags = 0;
    mxgad = gad = CreateGadget (MX_KIND, gad, &ng,
                                GTMX_TitlePlace, PLACETEXT_ABOVE,
                                GTMX_Labels, BassLines, GTMX_Scaled, TRUE,
                                GTMX_Spacing, gadfont->ta_YSize + 1, TAG_END);
```

```
    /* create a slider gadget */
    ng.ng_TopEdge = scr->Height - 170;
    ng.ng_Height = 10;
    ng.ng_Width = 100;
    ng.ng_GadgetText = "Left Justified";
    ng.ng_GadgetID = MYGAD_SLIDERLEFT;
    ng.ng_Flags = PLACETEXT_ABOVE;
    slid1gad = gad = CreateGadget (SLIDER_KIND, gad, &ng,
                                   GTSL_MaxLevelLen, 6,
                                   GTSL_MaxPixelLen, 64,
                                   GTSL_Max, 512,
                                   GTSL_Justification, GTJ_LEFT,
                                   GTSL_LevelFormat, "%lx",
                                   GTSL_LevelPlace, PLACETEXT_BELOW,
                                   TAG_END);

    /* create a slider gadget */
    ng.ng_TopEdge = ng.ng_TopEdge + 55;
    ng.ng_GadgetText = "Centered";
    ng.ng_GadgetID = MYGAD_SLIDERCENTER;
    slid2gad = gad = CreateGadget (SLIDER_KIND, gad, &ng,
                                   GTSL_MaxLevelLen, 6,
                                   GTSL_MaxPixelLen, 64,
                                   GTSL_Max, 512,
                                   GTSL_LevelFormat, "%lx",
                                   GTSL_Justification, GTJ_CENTER,
                                   GTSL_LevelPlace, PLACETEXT_BELOW,
                                   TAG_END);

    /* create a slider gadget */
    ng.ng_TopEdge = ng.ng_TopEdge + 55;
    ng.ng_GadgetText = "Right Justified";
    ng.ng_GadgetID = MYGAD_SLIDERRIGHT;
    slid3gad = gad = CreateGadget (SLIDER_KIND, gad, &ng,
                                   GTSL_MaxLevelLen, 6,
                                   GTSL_MaxPixelLen, 64,
                                   GTSL_Max, 512,
                                   GTSL_LevelFormat, "%lx",
                                   GTSL_Justification, GTJ_RIGHT,
                                   GTSL_LevelPlace, PLACETEXT_BELOW,
                                   TAG_END);

    /* Set these in case the listview cannot be created */
    ng.ng_LeftEdge = 240 + scr->WBorLeft;
    ng.ng_TopEdge = 30 + scr->WBorTop + (scr->Font->ta_YSize + 1);

    if (CreateDaList (BassSpaces))
    {
      /* create a listview gadget */
      ng.ng_LeftEdge = 240 + scr->WBorLeft;
      ng.ng_TopEdge = 30 + scr->WBorTop + (scr->Font->ta_YSize + 1);
      ng.ng_Width = 100;
      ng.ng_Height = 30 + gadfont->ta_YSize + 14;
      ng.ng_GadgetText = "Bass Spaces";
      ng.ng_GadgetID = MYGAD_LIST;
      listgad = gad = CreateGadget (LISTVIEW_KIND, gad, &ng,
                                    GTLV_ShowSelected, NULL,
                                    GTLV_Selected, 2,
                                    GTLV_MakeVisible, 2,
                                    GTLV_Labels, DaList,
                                    TAG_END);
    }

    /* create a checkbox gadget */
    ng.ng_LeftEdge = ng.ng_LeftEdge + 30;
    ng.ng_TopEdge = ng.ng_TopEdge + 130;
    ng.ng_Width = 50;
    ng.ng_Height = 30;
    ng.ng_GadgetText = "BigCheck";
    ng.ng_GadgetID = MYGAD_TEXT;
    ng.ng_Flags = PLACETEXT_ABOVE;
    checkgad = gad = CreateGadget (CHECKBOX_KIND, gad, &ng,
                                   GTCB_Scaled, TRUE, TAG_END);
```

```
    /* create a string gadget */
    ng.ng_LeftEdge = scr->Width - scr->WBorR
    ng.ng_TopEdge = ng.ng_TopEdge - 130;
    ng.ng_Width = 150;
    ng.ng_Height = gadfont->ta_YSize + 14;
    ng.ng_GadgetText = "String Immediate";
    ng.ng_GadgetID = MYGAD_STRING;
    stringgad = gad = CreateGadget (STRING_K
                                    GA_Immed

    /* create a text gadget */
    ng.ng_TopEdge = ng.ng_TopEdge + 55;
    ng.ng_Height = gadfont->ta_YSize + 2;
    ng.ng_GadgetText = "Echo After Click";
    ng.ng_GadgetID = MYGAD_TEXT;
    textgad = gad = CreateGadget (TEXT_KIND,
                                  GTTX_Borde
                                  GTTX_Front
                                  GTTX_BackP
                                  GTTX_Clipp
                                  TAG_END);

    /* create a palette gadget */
    ng.ng_TopEdge = ng.ng_TopEdge + 80;
    ng.ng_Width = 100;
    ng.ng_Height = 30;
    ng.ng_GadgetText = "Odd Colors";
    ng.ng_GadgetID = MYGAD_PAL;
    palgad = gad = CreateGadget (PALETTE_KIN
                                 GTPA_Color
                                 GTPA_NumCo
                                 GTPA_Color,
                                 GTPA_Indica
                                 TAG_END);

    /* create an integer gadget */
    ng.ng_Width = 150;
    ng.ng_LeftEdge = scr->Width + scr->WBorR
    ng.ng_TopEdge = scr->Height - 170;
    ng.ng_Height = gadfont->ta_YSize + 14;
    ng.ng_GadgetText = "Number, please";
    ng.ng_GadgetID = MYGAD_INT;
    intgad = gad = CreateGadget (INTEGER_KIN
                                 GTIN_MaxCha
                                 TAG_END);

    /* create a number gadget */
    ng.ng_TopEdge = ng.ng_TopEdge + 60;
    ng.ng_Width = 100;
    ng.ng_Height = gadfont->ta_YSize + 2;
    ng.ng_GadgetText = "Echo Number";
    ng.ng_GadgetID = MYGAD_NUM;
    numgad = gad = CreateGadget (NUMBER_KIND
                                 GTNM_FrontP
                                 GTNM_Format
                                 GTNM_Clippe
                                 GTNM_Number
                                 TAG_END);
  }
  else
    printf ("Could not create context\n");

  return (gad);
}

/*
 *  Create Menu.
 */

struct Menu *
CreateDaMenu (struct NewMenu *themenu, struc
{
  struct Menu *menusready;

  if (menusready = CreateMenus (themenu, TAG
```

```
>Width - scr->WBorRight - 190;
_TopEdge - 130;

t->ta_YSize + 14;
String Immediate";

eateGadget (STRING_KIND, gad, &ng,
            GA_Immediate, TRUE, TAG_END);


_TopEdge + 55;
t->ta_YSize + 2;
cho After Click";

eGadget (TEXT_KIND, gad, &ng,
          GTTX_Border, TRUE,
          GTTX_FrontPen, 3,
          GTTX_BackPen, 2,
          GTTX_Clipped, TRUE,
          TAG_END);


_TopEdge + 80;


dd Colors";

Gadget (PALETTE_KIND, gad, &ng,
         GTPA_ColorTable, colors,
         GTPA_NumColors, 7,
         GTPA_Color, 2,
         GTPA_IndicatorWidth, 21,
         TAG_END);



>Width + scr->WBorRight - 190;
Height - 170;
t->ta_YSize + 14;
umber, please";

Gadget (INTEGER_KIND, gad, &ng,
         GTIN_MaxChars, 14,
         TAG_END);


_TopEdge + 60;

t->ta_YSize + 2;
cho Number";

Gadget (NUMBER_KIND, gad, &ng,
         GTNM_FrontPen, 3,
         GTNM_Format, "%04ld",
         GTNM_Clipped, TRUE,
         GTNM_Number, 0xffff1,
         TAG_END);


eate context\n");









enu *thememu, struct VisualInfo *vi)


Menus (thememu, TAG_END))
```



```
    LayoutMenus (menusready, vi, GTMN_NewLookMenus, TRUE, TAG_E
  else
    printf ("Could not create menus\n");

  return (menusready);
}


/*
 *  Process menu events.
 */

BOOL
ProcessDaMenu (USHORT menunumber, struct Window * win, struct M
               struct VisualInfo * vi)
{
  USHORT menunum, itemnum, subnum;
  BOOL closeit = FALSE;
  menunum = MENUNUM (menunumber);
  itemnum = ITEMNUM (menunumber);
  subnum = SUBNUM (menunumber);

  switch (menunum)
  {
  case 0:                       /* project menu */
    closeit = TRUE;
    break;
  case 1:                       /* justify text menu */
    switch (itemnum)
    {
    case 0:                     /* left justify string */
      GT_SetGadgetAttrs (textgad, win, NULL,
                         GTTX_Justification, GTJ_LEFT,
                         GTTX_Text, "Left", TAG_END);
      break;
    case 2:                     /* center justify string */
      GT_SetGadgetAttrs (textgad, win, NULL,
                         GTTX_Justification, GTJ_CENTER,
                         GTTX_Text, "Center", TAG_END);
      break;
    case 4:                     /* right justify string */
      GT_SetGadgetAttrs (textgad, win, NULL,
                         GTTX_Justification, GTJ_RIGHT,
                         GTTX_Text, "Right", TAG_END);
    }
  }

  return (closeit);
}


/*
 *  Process IDCMP events
 *    Again, this is very incomplete.  You would do a lot more
 *    real application.
 */

void
ProcessEvents (struct Window *win, struct Menu *menustrip, stru
{
  struct IntuiMessage *msg;
  ULONG msgclass;
  LONG num;
  BOOL Closeflag = FALSE;
  STRPTR holdstring;
  USHORT menunumber;

  while (!Closeflag)
  {
    Wait (1 << win->UserPort->mp_SigBit);

    while ((!Closeflag) && (msg = GT_GetIMsg (win->UserPort)))
    {
      msgclass = msg->Class;
```

```
kMenus, TRUE, TAG_END);

dow * win, struct Menu * menustrip,

ext menu */

ify string */

stify string */

GTJ_CENTER,
, TAG_END);

tify string */

GTJ_RIGHT,
 TAG_END);

ould do a lot more if this were a

nu *menustrip, struct VisualInfo *vi)

(win->UserPort)))
```

```
      switch (msgclass)
      {
      case IDCMP_CLOSEWINDOW:
        Closeflag = TRUE;
        break;

      case IDCMP_GADGETDOWN:
        if (msg->IAddress == stringgad)
        {
          holdstring = ((struct StringInfo *) (stringgad->SpecialInfo))->Buffer;
          GT_SetGadgetAttrs (textgad, win, NULL,
                             GTTX_Text, holdstring, TAG_END);
        }
        break;

      case IDCMP_GADGETUP:
        if (msg->IAddress == intgad)
        {
          num = ((struct StringInfo *) (intgad->SpecialInfo))->LongInt;
          GT_SetGadgetAttrs (numgad, win, NULL, GTNM_Number, num,
                             GTNM_Justification, GTJ_RIGHT, TAG_END);
        }
        break;

      case IDCMP_MENUPICK:
        menunumber = msg->Code;
        while (menunumber != MENUNULL && !Closeflag)
        {
          Closeflag = ProcessDaMenu (menunumber, win, menustrip, vi);
          menunumber = (ItemAddress (menustrip, menunumber))->NextSelect;
        }
      }
      GT_ReplyIMsg (msg);
    }
  }
}

/*
 *  Free the memory used for the listview.
 */

void
FreeDaMemory (void)
{
  struct Node *freenode, *nextnode;

  freenode = (struct Node *) DaList->lh_Head;
  while (nextnode = (struct Node *) freenode->ln_Succ)
  {
    FreeMem (freenode, sizeof (struct Node));
    freenode = nextnode;
  }
  FreeMem (DaList, sizeof (struct List));
}


/*
 *  main().
 *    With SAS 6.0, you don't have to open libraries, so we can get straight
 *    to work.
 */

void
main (void)
{
  struct Screen *Gadscreen;
  struct Window *Gadwindow;
  struct VisualInfo *vi;
  struct TextFont *Appfont;
  struct Menu *menuptr;

  /* use a font you like */
  struct TextAttr nicefont =
  {
    "diamond.font",             /*  STRPTR  ta_Name     name of the font */
```

```
  15,                           /*  UWORD   ta_YSize    height of the font */
  FS_NORMAL,                    /*  UBYTE   ta_Style    intrinsic font style */
  FPF_DISKFONT                  /*  UBYTE   ta_Flags    font preferences and flags */
};

struct NewMenu Gadmenu[] =
{
  {NM_TITLE, "Project", 0, 0, 0, 0,},
  {NM_ITEM, "Quit", "Q", 0, 0, 0,},

  {NM_TITLE, "Justify Text", 0, 0, 0, 0,},
  {NM_ITEM, "Left", "L", 0, 0, 0,},
  {NM_ITEM, NM_BARLABEL, 0, 0, 0, 0,},
  {NM_ITEM, "Center", "M", 0, 0, 0,},
  {NM_ITEM, NM_BARLABEL, 0, 0, 0, 0,},
  {NM_ITEM, "Right", "R", 0, 0, 0,},
  {NM_END, NULL, 0, 0, 0, 0,},
};

if (Appfont = OpenDiskFont (&nicefont))
{
  if (Gadscreen = OpenScreenTags (NULL,
                                  SA_Left, 0,
                                  SA_Top, 0,
                                  SA_LikeWorkbench, TRUE,
                                  SA_Font, &nicefont,
                                  SA_Title, "Some New GadTools Features",
                                  TAG_DONE))
  {
    if ((vi = GetVisualInfo (Gadscreen, TAG_END)) != NULL)
    {
      if (CreateGads (Gadscreen, vi, &nicefont) != NULL)
      {
        if (menuptr = CreateDaMenu (Gadmenu, vi))
        {
          if (Gadwindow = OpenWindowTags (NULL,
                                          WA_Left, Gadscreen->LeftEdge,
                                    WA_Top, Gadscreen->TopEdge + Gadscreen->BarHeight,
                                  WA_Height, Gadscreen->Height - Gadscreen->BarHeight,
                                          WA_Width, Gadscreen->Width,
                                WA_MinWidth, Gadscreen->Height - Gadscreen->BarHeight,
                                          WA_MinHeight, Gadscreen->Width,
                                          WA_Gadgets, glist,
                                          WA_Title, "Lots of Gadgets",
                                          WA_CustomScreen, Gadscreen,
                                    WA_IDCMP, IDCMP_CLOSEWINDOW | IDCMP_ACTIVEWINDOW |
                                          IDCMP_GADGETDOWN | IDCMP_MENUPICK |
                                       IDCMP_GADGETHELP | IDCMP_GADGETUP | SLIDERIDCMP,
                                        WA_Flags, WFLG_DEPTHGADGET | WFLG_CLOSEGADGET |
                                          WFLG_ACTIVATE,
                                          WA_NewLookMenus, TRUE,
                                          WA_DragBar, TRUE,
                                          WA_CloseGadget, TRUE,
                                          TAG_DONE))
          {
            if (SetMenuStrip (Gadwindow, menuptr))
            {
              GT_RefreshWindow (Gadwindow, NULL);

              ProcessEvents (Gadwindow, menuptr, vi);

              ClearMenuStrip (Gadwindow);

              FreeMenus (menuptr);

              CloseWindow (Gadwindow);
            }
          }
        }
      }
      else
        printf ("Could not create gadgets\n");

      if (DaList)
        FreeDaMemory ();

        FreeGadgets (glist);

        FreeVisualInfo (vi);
      }

      else
        printf ("Could not get visual info\n

      CloseScreen (Gadscreen);
    }
    else
      printf ("Could not open screen\n");

    CloseFont (Appfont);
  }
  else
    printf ("Could not open %s\n", nicefont
}
```

```
t get visual info\n");

open screen\n");

en %s\n", nicefont.ta_Name);
```