```
;/*
lc -b1 -cfist -v -y -j73 relative
blink FROM LC:/lib/c.o relative.o TO relative LIB LIB:lc.lib LIB:amiga.lib
quit
*/
/*
 * relative.c - shows custom gadget relativity
 *
 * (c) Copyright 1992 Commodore-Amiga, Inc.  All Rights Reserved.
 * Preliminary and Confidential.
 *
 * custom gadget relativity allows a gadget to arbitrarily resize
 * itself whenever the window changes size.  This is a complete superset
 * of the functionality of the old GRELWIDTH, GRELRIGHT, etc., features.
 * This example shows a subclass of gadgetclass whose imagery comes
 * from frameiclass.  This gadget's property is that it is always half
 * the size of its domain, and centered within it.  That is, it's always
 * half as wide and half as tall as the inner area of its window, and
 * centered within that area.
 */

#include <intuition/intuition.h>
#include <intuition/gadgetclass.h>
#include <intuition/imageclass.h>
#include <intuition/cghooks.h>
#include <intuition/classes.h>
#include <intuition/classusr.h>

#include <clib/alib_protos.h>
#include <clib/alib_stdio_protos.h>
#include <clib/exec_protos.h>
#include <clib/graphics_protos.h>
#include <clib/intuition_protos.h>

struct Library *GfxBase = NULL;
struct Library *IntuitionBase = NULL;
struct Window *win = NULL;
struct Gadget *gad = NULL;
Class *customrelclass = NULL;

Class *initCustomRelClass(void);
ULONG __saveds __asm dispatchCustomRel( register __a0 Class *cl,
                                         register __a2 Object *o,
                                         register __a1 Msg msg );
void                renderCustomRel( struct Gadget *gad,
                                   struct RastPort *rp,
                                   struct GadgetInfo *gi );
void                layoutCustomRel( struct Gadget *gad,
                                   struct GadgetInfo *gi,
                                   ULONG initial );
LONG                handleCustomRel( struct Gadget *gad,
                                   struct gpInput *msg );

void main(void)
{
  if ( GfxBase = OpenLibrary("graphics.library",39) )
  {
    if ( IntuitionBase = OpenLibrary("intuition.library",39) )
    {
      if ( customrelclass = initCustomRelClass() )
      {
        if ( gad = NewObject( customrelclass, NULL,
            GA_Left, 20,
            GA_Top, 20,
            GA_Width, 20,
            GA_Height, 20,
            GA_RelVerify, TRUE,
            GA_Immediate, TRUE,
            TAG_DONE ) )
        {
          if ( win = OpenWindowTags( NULL,
              WA_Title, "Custom Relativity Demo",
              WA_CloseGadget, TRUE,
              WA_DepthGadget, TRUE,
              WA_DragBar, TRUE,
              WA_SizeGadget, TRUE,
              WA_Gadgets, gad,
              WA_Activate, TRUE,
              WA_IDCMP, IDCMP_GADGETHELP | IDCMP_RAWKEY | IDCMP_CLOSEWINDOW |
                        IDCMP_GADGETDOWN | IDCMP_GADGETUP,
              WA_Width, 150,
              WA_Height, 150,
              WA_MinWidth, 50,
              WA_MinHeight, 50,
              WA_MaxWidth, ~0,
              WA_MaxHeight, ~0,
              WA_NoCareRefresh, TRUE,
              TAG_DONE ) )
          {
            BOOL terminated = FALSE;
            struct IntuiMessage *imsg;

            /* Turn on Gadget Help */
            HelpControl( win, HC_GADGETHELP );

            while (!terminated)
            {
              Wait (1 << win->UserPort->mp_SigBit);
              while (imsg = (struct IntuiMessage *) GetMsg(win->UserPort))
              {
                switch ( imsg->Class )
                {
                case IDCMP_CLOSEWINDOW:
                  terminated = TRUE;
                  break;

                case IDCMP_RAWKEY:
                  printf("RAWKEY %lx\n", imsg->Code);
                  break;

                case IDCMP_GADGETUP:
                  printf("Gadget Up\n");
                  break;

                case IDCMP_GADGETDOWN:
                  printf("Gadget Down\n");
                  break;

                case IDCMP_GADGETHELP:
                  if ( imsg->IAddress == NULL )
                  {
                    printf("Gadget Help: Mouse not over window\n");
                  }
                  else if ( imsg->IAddress == (APTR) win )
                  {
                    printf("Gadget Help: Mouse in window, not over a gadget\n");
                  }
                  else
                  {
                    /* Detect system gadgets.  Figure out by looking at
                     * system-gadget-type bits in GadgetType field:
                     */
                    LONG sysgtype =
                      ((struct Gadget *)imsg->IAddress)->GadgetType & GTYP_SYSTYPEMASK;
                    switch ( sysgtype )
                    {
                    case GTYP_SIZING:
                      printf("Gadget Help for window sizing gadget\n");
                      break;

                    case GTYP_WDRAGGING:
                      printf("Gadget Help for window drag-bar\n");
                      break;

                    case GTYP_WUPFRONT:
                      printf("Gadget Help for window depth gadget\n");
                      break;

                    case GTYP_WDOWNBACK:
                      printf("Gadget Help for window zoom gadget\n");
```

```
                      break;

                    case GTYP_CLOSE:
                      printf("Gadget Help for window close gadget\n");
                      break;

                    case 0:
                      /* In this example, we only have one gadget,
                       * so we know which one it is.  Normally, you'd
                       * have to figure that out here, using the
                       * usual techniques you already use for other
                       * gadget messages.
                       */
                      printf("Gadget Help for gadget, code 0x%x\n",
                          imsg->Code);
                      break;

                    default:
                      printf("Gadget Help on some other system gadget\n");
                      break;
                    }
                  }
                }
                ReplyMsg((struct Message *)imsg);
              }
            }
            CloseWindow( win );
          }
          DisposeObject( gad );
        }
        FreeClass( customrelclass );
      }
      CloseLibrary( IntuitionBase );
    }
    CloseLibrary( GfxBase );
  }
}


/* initCustomRelClass()
 *
 * Initialize a simple private subclass of gadgetclass that
 * knows about GM_LAYOUT.
 *
 */

Class *initCustomRelClass( void )
{
  Class *cl;

  /* Create a private class: */
  if ( cl = MakeClass( NULL, "gadgetclass", NULL, 0, 0 ) )
  {
    cl->cl_Dispatcher.h_SubEntry = NULL;
    cl->cl_Dispatcher.h_Entry = dispatchCustomRel;
    cl->cl_Dispatcher.h_Data = NULL;
  }
  return ( cl );
}

/* dispatchCustomRel()
 *
 * boopsi dispatcher for the custom relativity class.
 *
 */

ULONG __saveds __asm
dispatchCustomRel( register __a0 Class *cl,
                    register __a2 Object *o,
                    register __a1 Msg msg )
{
  ULONG retval = 1;
  Object *newobj;

  switch ( msg->MethodID )
  {
  case OM_NEW:
    if ( retval = (ULONG)(newobj = (Object *) DoSuperMethodA( cl, o, msg )) )
    {
      /* Set custom relativity */
      ((struct Gadget *)newobj)->Flags |= GFLG_RELSPECIAL;

      /* Tell Intuition this gadget supports gadget help */
      SetAttrs(newobj, GA_GadgetHelp, TRUE, TAG_DONE, NULL);

      /* Attempt to allocate a frame.  If I can't, then
       * delete myself and fail.
       */
      if ( ! ( ((struct Gadget *)newobj)->GadgetRender =
          NewObject( NULL, "frameiclass",
                     IA_FrameType, FRAME_BUTTON,
                     TAG_DONE ) ) )
      {
        CoerceMethod( cl, o, OM_DISPOSE );
        retval = NULL;
      }
    }
    break;

  case GM_LAYOUT:
    layoutCustomRel( (struct Gadget *)o, ((struct gpLayout *)msg)->gpl_GInfo,
        ((struct gpLayout *)msg)->gpl_Initial );
    break;

  case GM_RENDER:
    renderCustomRel( (struct Gadget *)o, ((struct gpRender *) msg)->gpr_RPort,
        ((struct gpRender *) msg)->gpr_GInfo );
    break;

  case GM_GOACTIVE:
    return( GMR_MEACTIVE );
    break;

  case GM_HELPTEST:
    return(GMR_HELPCODE | 0x0000C0DE);
    break;

  case GM_HANDLEINPUT:
    retval = handleCustomRel( (struct Gadget *)o, (struct gpInput *)msg );
    break;

  case OM_DISPOSE:
    DisposeObject( ((struct Gadget *)o)->GadgetRender );
    /* fall through to default */
  default:
    retval = (ULONG) DoSuperMethodA( cl, o, msg );
  }
  return( retval );
}


/* renderCustomRel()
 *
 * Simple routine to draw my imagery based on my selected state.
 *
 */
void
renderCustomRel( struct Gadget *gad, struct RastPort *rp, struct GadgetInfo *gi )
{
  DrawImageState( rp, gad->GadgetRender, gad->LeftEdge, gad->TopEdge,
      (gad->Flags & GFLG_SELECTED) ? IDS_SELECTED : IDS_NORMAL,
      gi ? gi->gi_DrInfo : NULL );
}


/* layoutCustomRel()
 *
 * Lay myself out based on my domain dimensions.  Refigure my own size,
 * then inform my image of the size change.
 *
```

```
 */
void
layoutCustomRel( struct Gadget *gad, struct GadgetInfo *gi, ULONG initial )
{
  if ( gi->gi_Requester )
  {
    /* Center it within the requester */
    gad->Width = gi->gi_Domain.Width / 2;
    gad->Height = gi->gi_Domain.Height / 2;
    gad->LeftEdge = gad->Width / 2;
    gad->TopEdge = gad->Height / 2;
  }
  else
  {
    /* Center it within the window, after accounting for
     * the window borders
     */
    gad->Width = ( gi->gi_Domain.Width -
        gi->gi_Window->BorderLeft - gi->gi_Window->BorderRight ) / 2;
    gad->Height = ( gi->gi_Domain.Height -
        gi->gi_Window->BorderTop - gi->gi_Window->BorderBottom ) / 2;
    gad->LeftEdge = ( gad->Width / 2 ) + gi->gi_Window->BorderLeft;
    gad->TopEdge = ( gad->Height / 2 ) + gi->gi_Window->BorderTop;
  }
  SetAttrs( gad->GadgetRender,
    IA_Width, gad->Width,
    IA_Height, gad->Height,
    TAG_DONE );
}


/* handleCustomRel()
 *
 * Routine to handle input to the gadget.  Behaves like a basic
 * hit-select gadget.
 *
 */
LONG
handleCustomRel( struct Gadget *gad, struct gpInput *msg )
{
  WORD selected = 0;
  struct RastPort *rp;
  LONG retval = GMR_MEACTIVE;

  /* Could send IM_HITTEST to image instead */
  if ( ( msg->gpi_Mouse.X >= 0 ) &&
      ( msg->gpi_Mouse.X < gad->Width ) &&
      ( msg->gpi_Mouse.Y >= 0 ) &&
      ( msg->gpi_Mouse.Y < gad->Height ) )
  {
    selected = GFLG_SELECTED;
  }

  if ((msg->gpi_IEvent->ie_Class == IECLASS_RAWMOUSE) &&
      (msg->gpi_IEvent->ie_Code == SELECTUP))
  {
    /* gadgetup, time to go */
    if ( selected )
    {
      retval = GMR_NOREUSE | GMR_VERIFY;
    }
    else
    {
      retval = GMR_NOREUSE;
    }
    /* and unselect the gadget on our way out... */
    selected = 0;
  }

  if ( ( gad->Flags & GFLG_SELECTED ) != selected )
  {
    gad->Flags ^= GFLG_SELECTED;
    if ( rp = ObtainGIRPort( msg->gpi_GInfo ) )
    {
      DoMethod( (Object *)gad, GM_RENDER, msg->gpi_GInfo, rp, GREDRAW_UPDATE );
      ReleaseGIRPort( rp );
    }
  }

  return( retval );
}
```