

```

/* cliptext.c - Execute me to compile me with Lattice 5.10a
LC -cfistq -v -y -j73 cliptext.c
Blink FROM LIB:c.o,cliptext.o TO cliptext LIBRARY LIB:LC.lib,LIB:Amiga.lib
quit

```

Sept. 17 1991 by John Orr
*/

```

#include <exec/types.h>
#include <dos/rdargs.h>
#include <dos/dosextens.h>
#include <intuition/intuition.h>
#include <graphics/text.h>
#include <graphics/displayinfo.h>
#include <graphics/regions.h>
#include <graphics/gfx.h>
#include <libraries/diskfont.h>
#include <utility/tagitem.h>
#include <clib/exec_protos.h>
#include <clib/dos_protos.h>
#include <clib/layers_protos.h>
#include <clib/alib_stdio_protos.h>
#include <clib/intuition_protos.h>
#include <clib/graphics_protos.h>
#include <clib/diskfont_protos.h>

```

```

#ifdef LATTICE
int CXBRK(void) { return(0); } /* Disable Lattice CTRL/C handling */
int chkabort(void) { return(0); }
#endif

```

```

UBYTE      *vers = "\0$VER: cliptext 36.5";

```

```

#define BUFSIZE      4096
#define FONT_NAME    0
#define FONT_SIZE    1
#define FILE_NAME     2
#define JAM_MODE     3
#define XASP         4
#define YASP         5
#define NUM_ARGS     6

```

```

#define DEFAULTFONTSIZE 11L
#define DEFAULTJAMMODE  0L
#define DEFAULTXASP     0L
#define DEFAULTYASP     0L

```

```

void      MainLoop(void);

```

```

LONG      args[NUM_ARGS];
struct TagItem tagitem[2];

```

```

UBYTE      buffer[BUFSIZE];
BPTR       myfile;

```

```

struct Window *mywin;
struct RastPort *myrp;

```

```

struct TTextAttr myta;
struct TextFont *myfont;
struct Rectangle myrectangle;

```

```

struct Region *new_region;

```

```

struct Library *DiskfontBase, *IntuitionBase, *LayersBase, *GfxBase;
struct IntuiMessage *mymsg;
struct DrawInfo *mydrawinfo;

```

```

void
main(int argc, char **argv)
{

```

```

    struct RDArgs *myrda;
    struct DisplayInfo mydi;
    ULONG      mymodeid;

```

```

    LONG      mydefaultfontsize = DEFAULTFONTSIZE;
    LONG      mydefaultJAMMode = DEFAULTJAMMODE;

```

```

LONG      mydefaultXASP = 0L;
LONG      mydefaultYASP = 0L;

```

```

args[FONT_NAME] = (LONG) "topaz.font";
args[FONT_SIZE] = (LONG) &mydefaultfontsize;
args[FILE_NAME] = (LONG) "s:startup-sequence";
args[JAM_MODE] = (LONG) &mydefaultJAMMode;
args[XASP] = (LONG) &mydefaultXASP;
args[YASP] = (LONG) &mydefaultYASP;

```

```

/*
 * dos.library standard command line parsing - see the article "Standard
 * Command Line Parsing" from the May/June 1991 issue of Amiga Mail, or see
 * the AmigaDOS Manual for more details
 */

```

```

if (myrda = ReadArgs("FontName,FontSize/N,FileName,Jam/N,XASP/N,YASP/N\n",
    args, NULL))
{

```

```

    /* open the file to display */
    if (myfile = Open((UBYTE *) args[FILE_NAME], MODE_OLDFILE))
    {
        /* open the libraries */
        if (DiskfontBase = OpenLibrary("diskfont.library", 36L))
        {
            if (IntuitionBase = OpenLibrary("intuition.library", 36L))
            {
                if (GfxBase = OpenLibrary("graphics.library", 36L))
                {
                    if (LayersBase = OpenLibrary("layers.library", 36L))
                    {
                        /* Open that window */
                        if (mywin = OpenWindowTags(NULL,
                            WA_MinWidth, 100,
                            WA_MinHeight, 100,
                            WA_SmartRefresh, TRUE,
                            WA_SizeGadget, TRUE,
                            WA_CloseGadget, TRUE,

```

```

                            /*
                             * This application wants to hear about three things: When the
                             * user clicks the window's close gadget, when the user starts to
                             * resize the window, and when the user has finished resizing the
                             * window.
                             */
                            WA_IDCMP, IDCMP_CLOSEWINDOW | IDCMP_NEWSIZE | IDCMP_SIZEVERIFY,
                            WA_DragBar, TRUE,
                            WA_DepthGadget, TRUE,
                            WA_Title, (ULONG) args[FILE_NAME],
                            TAG_END))

```

```

        {
            tagitem[0].ti_Tag = TA_DeviceDPI;

```

```

        /*
         * see if there is a non-zero value in the XASP or YASP fields.
         * Diskfont.library will get very upset (divide by zero GURU)
         * if you give it a zero XDPI or YDPI value.
         */

```

```

        /* if there is a zero value in one of them... */
        if (((*(ULONG *) args[XASP]) == 0) || (((*(ULONG *) args[YASP]) == 0))
        {

```

```

            /*
             * ...use the aspect ratio of the current display as a
             * default...
             */

```

```

            mymodeid = GetVPMODEID(&(mywin->WScreen->ViewPort));
            if (GetDisplayInfoData(NULL, (UBYTE *) &mydi,
                sizeof(struct DisplayInfo), DTAG_DISP, mymodeid))
            {
                mydefaultXASP = mydi.Resolution.x;
                mydefaultYASP = mydi.Resolution.y;

```

```

            /*
             * notice that the X and Y get _swapped_ to keep the look
             * of the font glyphs the same using screens with different
             * aspect ratios.

```

```

        /*
        * args[YASP] = (LONG) & mydefaultYASP;
        * args[XASP] = (LONG) & mydefaultYASP;
        }
        else
            /* ...unless something is preventing us from
            * getting the screens resolution. In that
            * case, forget about the DPI tag. */
            tagitem[0].ti_Tag = TAG_END;
    }

    /*
    * Here we have to put the X and Y DPI into the TA_DeviceDPI
    * tag's data field. THESE ARE NOT REAL X AND Y DPI VALUES FOR
    * THIS FONT OR THE DISPLAY. They only serve to supply the
    * diskfont.library with values to calculate the aspect ratio.
    * The X value gets stored in the upper word of the tag value
    * and the Y DPI gets stored in the lower word. Because
    * ReadArgs() stores the _address_ of integers it gets from the
    * command line, you have to dereference the pointer it puts
    * into the argument array, which results in some ugly casting.
    */
    tagitem[0].ti_Data =
        (ULONG) (((UWORD) * ((ULONG *) args[XASP]) << 16) |
        (UWORD) * ((ULONG *) args[YASP])));
    tagitem[1].ti_Tag = TAG_END;

    /*
    * set up the TTextAttr structure to match the font the user
    * requested.
    */
    myta.tta_Name = (STRPTR) args[FONT_NAME];
    myta.tta_Ysize = *((LONG *) args[FONT_SIZE]);
    myta.tta_Style = FSP_TAGGED;
    myta.tta_Plags = 0L;
    myta.tta_Tags = tagitem;

    /* open that font */
    if (myFont = OpenDiskFont(kmyta))
    {
        /*
        * This is for the layers.library clipping region that gets
        * attached to the window. This prevents the application
        * from unnecessarily rendering beyond the bounds of the
        * inner part of the window. For now, you can ignore the
        * layers stuff if you are just interested in learning about
        * using text. For more information on clipping regions and
        * layers, see the Layers chapter of the RKM:libraries
        * manual.
        */
        myrectangle.MinX = mywin->BorderLeft;
        myrectangle.MinY = mywin->BorderTop;
        myrectangle.MaxX = mywin->Width - (mywin->BorderRight + 1);
        myrectangle.MaxY = mywin->Height - (mywin->BorderBottom + 1);

        /* more layers stuff */
        if (new_region = NewRegion())
        {
            /* Even more layers stuff */
            if (OrRectRegion(new_region, kmyrectangle))
            {
                InstallClipRegion(mywin->WLayer, new_region);
            }
        }

        /*
        * obtain a pointer to the window's rasterport and set up
        * some of the rasterport attributes. This example obtains
        * the text pen for the window's screen using the
        * GetScreenDrawInfo() function.
        */
        myrtp = mywin->RPort;
        SetFont(myrtp, myfont);
        if (mydrawinfo = GetScreenDrawInfo(mywin->WScreen))
        {
            SetApPen(myrtp, mydrawinfo->dr_Pen1(TEXTPEN));
            FreeScreenDrawInfo(mywin->WScreen, mydrawinfo);
        }
        SetDPMd(myrtp, (BYTE) (*((LONG *) args[AM_MODE])));
    }

```

```

        }
        }
        }
        DisposeRegion(new_region);
        CloseFont(myFont);
        CloseWindow(mywin);
        CloseLibrary(layersBase);
        CloseLibrary(GfxBase);
        CloseLibrary(IntuitionBase);
        CloseLibrary(DiskFontBase);
        Close(myfile);
        FreeArgs(myrda);
    }
    else
        VPrintf("Error parsing arguments\n", NULL);
}

void
MainLoop(void)
{
    LONG    count, actual, position;
    BOOL    ack = TRUE, waitfornewsz = FALSE;
    struct Task *mytask;

    mytask = FindTask(NULL);
    Move(myrtp, mywin->BorderLeft + 1, mywin->BorderTop + myFont->tf_Ysize + 1);

    /* while there's something to read, fill the buffer */
    while (((actual = Read(myfile, buffer, BUFSIZE)) > 0) && ack)
    {
        position = 0;
        count = 0;

        while (position <= actual)
        {
            if (!waitfornewsz)
            {
                while ( ( (buffer[count] >= myFont->tf_LoChar) &&
                (buffer[count] <= myFont->tf_HiChar) ) &&
                (count <= actual) )
                {
                    count++;
                }
            }
            if (buffer[count] == 0x0A)
                Move(myrtp, mywin->BorderLeft, myrtp->cp_y + myFont->tf_Ysize + 1);
            count++;
            position = count;
        }
        else
            WaitPort(mywin->UserPort);

        while (mymsg = (struct IntuiMessage *) GetMsg(mywin->UserPort))
        {
            /* The user clicked the close gadget */
            if (mymsg->Class == IDCMP_CLOSEWINDOW)
            {
                ack = FALSE;
                position = actual + 1;
                ReplyMsg((struct Message *) mymsg);
            }
            /* The user picked up the window's sizing gadget */
            else if (mymsg->Class == IDCMP_SIZEVERIFY)
            {

```

```

/*
 * When the user has picked up the window's sizing gadget when the
 * IDCMP_SIZEVERIFY flag is set, the application has to reply to this
 * message to tell Intuition to allow the user to move the gadget and
 * resize the window. The reason for using this here is because of
 * the nature of the Text() routine (and most other functions that
 * use the blitter). The Text() function only asks the blitter to
 * perform some rendering, it does not wait for the blitter to finish
 * the rendering. Some time can lapse between the finish of the
 * Text() call and the finish of the corresponding blitter operation.
 * In this period, the user can resize the window so that the display
 * area is smaller than it was when the blitter request was made. The
 * blitter will do its rendering according to the original size of
 * the window, which can cause the blitter to write into the window's
 * borders. To prevent this, when the user attempts to resize the
 * window, we have to wait for the blitter to finish any outstanding
 * blitter operations before we allow the window to be resized.
 */

/*
 * if this example had instead asked to hear about IDCMP events that
 * could take place between SIZEVERIFY and NEWSIZE events (especially
 * INTUITICKS), it should turn off those events here using
 * ModifyIDCMP().
 */

/*
 * After we allow the user to resize the window, we cannot write into
 * the window until the user has finished resizing it because we need
 * the window's new size to adjust the clipping area. Specifically,
 * we have to wait for an IDCMP_NEWSIZE message which Intuition will
 * send when the user lets go of the resize gadget. For now, we set
 * the waitfornewsize flag to stop rendering until we get that
 * NEWSIZE message.
 */

waitfornewsize = TRUE;
WaitBlit();

/* The blitter is done, let the user resize the window */
ReplyMsg((struct Message *) mymsg);
}
else
{
    ReplyMsg((struct Message *) mymsg);
    waitfornewsize = FALSE;

    /*
     * the user has resized the window, so get the new window dimensions
     * and readjust the layers clipping region accordingly.
     */
    myrectangle.MinX = mywin->BorderLeft;
    myrectangle.MinY = mywin->BorderTop;
    myrectangle.MaxX = mywin->Width - (mywin->BorderRight + 1);
    myrectangle.MaxY = mywin->Height - (mywin->BorderBottom + 1);
    InstallClipRegion(mywin->WLayer, NULL);
    ClearRegion(new_region);
    if (OrRectRegion(new_region, &myrectangle))
        InstallClipRegion(mywin->WLayer, new_region);
    else
    {
        aok = FALSE;
        position = actual + 1;
    }
}
}
/* check for user break */
if (mytask->tc_SigRecvd & SIGBREAKF_CTRL_C)
{
    aok = FALSE;
    position = actual + 1;
}
}

/*
 * if we reached the bottom of the page, clear the rastport and move back
 * to the top
 */

```

```

/*
if (myrp->cp_y > (mywin->Height - (mywin->BorderBottom + 2)))
{
    Delay(25);

    /*
     * Set the entire rastport to color zero. This will not overwrite the
     * window borders because of the layers clipping.
     */
    SetRast(myrp, 0);
    Move(myrp,
        mywin->BorderLeft + 1,
        mywin->BorderTop + myfont->tf_YSize + 1);
}
}
if (actual < 0)
    VPrintf("Error while reading\n", NULL);
}

```

