

```

    args[YASP] = (LONG) & mydefaultXASP;
    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 = FSF_TAGGED;
myta.tta_Flags = 0L;
myta.tta_Tags = tagitem;

/* open that font */
if (myfont = OpenDiskFont(&myta))
{
    /*
     * 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, &myrectangle))
        {
            InstallClipRegion(mywin->WLayer, new_region);
        }

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

```

```

        MainLoop();
    }
    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          aok = TRUE, waitfornewsize = FALSE;
    struct Task   *mytask;

    mytask = FindTask(NULL);
    Move(myrp, 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) && aok)
    {
        position = 0;
        count = 0;

        while (position <= actual)
        {
            if (!(waitfornewsize))
            {
                while ( ( (buffer[count] >= myfont->tf_LoChar) &&
                          (buffer[count] <= myfont->tf_HiChar) ) &&
                        (count <= actual) )
                    count++;

                Text(myrp, &(buffer[position]), (count) - position);

                while ( ( (buffer[count] < myfont->tf_LoChar) ||
                          (buffer[count] > myfont->tf_HiChar) ) &&
                        (count <= actual) )
                {
                    if (buffer[count] == 0x0A)
                        Move(myrp, mywin->BorderLeft, myrpp->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)
                {
                    aok = 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
 * sizing gadget and resize the window. The reason for using this
 * here is because the user can resize the window while cliptext.c
 * is rendering text to the window. Cliptext.c has to stop rendering
 * text when it receives an IDCMP_SIZEVERIFY message.
 */

/*
 * 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 (myxp->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);
}
}
}

```

