ESV Workstation

1.3.1 Release Notes

Part Number

517970-105

EVANS & SUTHERLAND COMPUTER CORPORATION
Salt Lake City, Utah

ESV Workstation

1.3.1 Release Notes

EVANS & SUTHERLAND COMPUTER CORPORATION
Salt Lake City, Utah

DOCUMENTATION WARRANTY:

PURPOSE: This documentation is provided to assist an Evans & Sutherland trained BUYER in using a product purchased from Evans & Sutherland. It may contain errors or omissions that only a trained individual may recognize. Changes may have occurred to the hardware/software, to which this documentation refers, which are not included in this documentation or may be on a separate errata sheet. Use of this documentation in such changed hardware/software could result in damage to hardware/software. User assumes full responsibility of all such results of the use of this data.

WARRANTY: This document is provided, and Buyer accepts such documentation, "AS-IS" and with "ALL FAULTS, ERRORS, AND OMISSIONS". BUYER HEREBY WAIVES ALL IMPLIED AND OTHER WARRANTIES, GUARANTIES, CONDITIONS OR LIABILITIES, EXPRESSED OR IMPLIED ARISING BY LAW OR OTHERWISE, INCLUDING, WITHOUT LIMITATIONS, ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. BUYER FURTHER HOLDS SELLER HARMLESS OF ANY DIRECT OR INDIRECT DAMAGES, INCLUDING CONSEQUENTIAL DAMAGES.

Part Number: 517970-105 AA
December, 1990

Table of Contents

1. Overview

The 1.3.1 Release adds the Overlay/Under planes and PHIGS stereo functionality to the ESV Workstation. The *1.3.1 Release Notes* are arranged as follows:

- Chapter 1, "Overview" (this chapter) describes the X.11 Overlay Extension and PHIGS stereo implementation.
- Chapter 2, "Known Bugs in the System," describes the ES/PSX and ES/PEX bugs that have been fixed in the 1.3.1 Release.
- Chapter 3, "Documentation Corrections," describes additions and corrections to the ESV Workstation documentation.
- Chapter 4, "Installation Instructions," describes the installation procedure for the 1.3.1 Release tape.
- Appendix A, "PHIGS Stereo Example Program," contains the code listing for a PHIGS stereo example which is included on the 1.3.1 Release tape.

X.11 Overlay Extension

The X.11 Overlay Extension provides easy access to the ESV Workstation overlay hardware. An example program that demonstrates the use of overlay planes on the ESV Workstation will be found in **/usr/people/fstest/demo/overlay.c.**

XAllocOverlayPlanes

```
Status
XAllocOverlayPlanes(dpy, win, planes)
        Display *dpy;
        Window win;
        unsigned int planes;
```

Allocate overlay planes for a specific window.

On the ESV Workstation, the total number of overlay planes is 4. These planes are shared with planes that identify PHIGS workstations. Each plane allocated for overlay reduces the total number of PHIGS workstations by one-half. If all 4 are allocated for overlay, a maximum number of 12 PHIGS workstations will be available. They can be allocated for overlay from 1 to 4.

If too many planes are requested or no planes can be allocated, a **BadValue** or **BadAlloc** error is returned.

Valid pixel values for the window will be 0 to (2^n-1), where **n** is the number of overlay planes.

XFreeOverlayPlanes

```
Status
XFreeOverlayPlanes(dpy, win)
        Display *dpy;
        Window win;
```

Free all the overlay planes allocated for this window. This makes the resource available for other applications.

XStoreOverlayColor

```
Status
XStoreOverlayColor(dpy, win, colorcell)
        Display *dpy;
        Window win;
        XColor *colorcell;
```

Set the color to be displayed by this pixel value in the overlay or colormap associated with a window. This routine will return **BadValue** if the color of the pixel cannot be changed or the pixel is not a valid pixel for this window.

Storing the color does not change the color displayed on the screen. To actually change the color, you have to install the overlay colormap.

Pixel value 0 (zero) is the transparent overlay color.

XInstallOverlayColormap

```
Status
XInstallOverlayColormap(dpy, win)
        Display *dpy;
        Window win;
```

Load the overlay or colormap for a window into the hardware colormap.

XSelectLayer

```
Status
XSelectLayer(dpy, win, layer)
        Display *dpy;
        Window win;
        int layer;
```

Select the layer in which following graphics commands will be done. Layer 0 is overlay, 1 is the normal frame buffer.

Restrictions

- ESV does not support ALU operation in the overlay planes. This means that any X operation with an ALU mode other than **GXcopy** cannot be done.
- Any X operation that needs both a foreground and a background color will not work correctly. The foreground color will be written, but the background will not.
- **XGetImage** will return the pixels in the currently selected layer. The overlay image will be returned in the low order bits of 32 bit pixels. **XGetImage** does not work with overlay planes in the 1.3.1 Release.
- **XPutImage** will work when overlay layers are selected. The image must be stored in the low order bits of 32 bit pixels. **XPutImage** does not work with overlay planes in the 1.3.1 Release.

PHIGS Stereo Implementation

The ESV Workstation provides a method for creating stereo images through PHIGS. This method includes two separate "screens" in the X server, one for monoscopic applications and one for stereoscopic applications. The stereo application is responsible for setting up the stereoscopic fields in the view table and for opening the X connection to the stereo screen. Everything else is handled by the X server.

Interface Overview

The ESV Workstation contains a special screen in the X server to support stereoscopic applications. The stereo application must open a window somewhere in the "stereo screen" (**host:0.1**). The user toggles between screens by moving the cursor off the left or right side of the screen.

The stereo screen is 512 pixels high. If PHIGS graphics are displayed in this window, the system will traverse the graphics structure twice to display both stereoscopic fields.

Using the PHIGS Viewing Model for Stereo

The stereo application must create the left-eye and right-eye stereoscopic fields in the PHIGS view table. These views are used by the system during traversal. To tell the system which stereoscopic field should be used for each eye, a PHIGS **GSE** is provided, which is an extension of the **SET_VIEW_INDEX** element. Rather than indicating a single index in the GSE, the application indicates three indices: one for left-eye, one for right-eye, and one for monoscopic viewing.

The left-eye and right-eye views are established in the View Reference Coordinate (VRC) system in VRC coordinates, as shown in figure 1. Left and right are established by shifting the Projection Reference Point (PRP) off the VRC *z*-axis in the *x*-direction. Note that the viewing frustum is created by passing planes through the PRP and the corners of the viewing window, which is a rectangle on the viewing plane.

The closer the PRP is to the viewing plane, the more severe the perspective angle. The view window rectangle is specified by the **window** field in the **Pviewmapping3** structure that is passed to **pevalviewmappingmatrix3**.

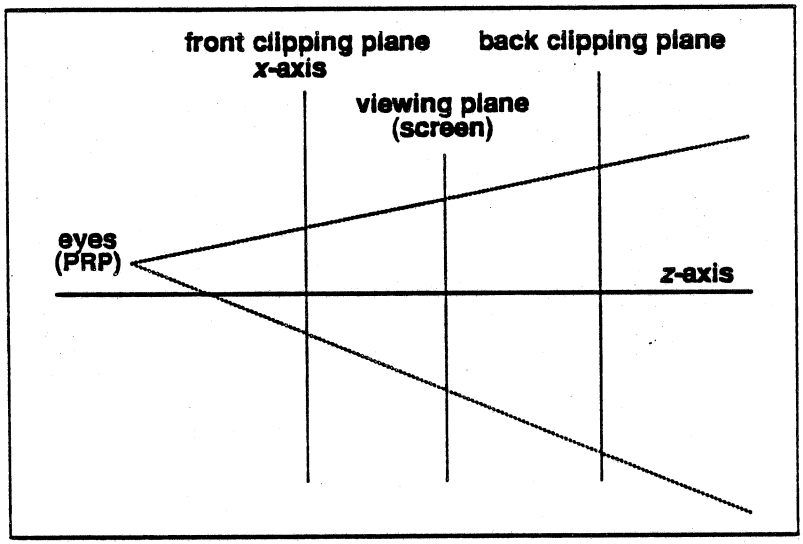

Figure 1. View Reference Coordinate System

The viewing plane coincides with the surface of the monitor screen, and the viewing window corresponds to the X window on the screen in which the graphics will be displayed. The following parameters should be carefully selected to create the best stereo image:

- PRP *z*-value (the distance between eyes and the monitor screen),
- The size of the X window in which the 3D object is drawn, and
- PRP *x*-value (the interocular separation).

In the beginning of the example program in "Appendix B", you will see a constant, **INCHES_TO_VRC**, which allows you to map physical inches to VRC distances. It is critical that real inches are correctly converted to units of VRC space, so that the stereoscopic fields are correctly calculated.

```
/* Stereo params in physical inches */
#define EYE_TO_SCREEN    36.0
#define WINDOW_SIZE      9.0
#define VIEW_RATIO       (EYE_TO_SCREEN / WINDOW_SIZE)
/* Stereo params in VRC units */
#define VRC_WINDOW_SIZE 4.0
#define INCHES_TO_VRC    (VRC_WINDOW_SIZE / WINDOW_SIZE)
#define VRC_PRP          (EYE_TO_SCREEN * INCHES_TO_VRC)
#define HALF_OCCULAR     (1.25 * INCHES_TO_VRC)
```

Stereo Setup Procedure

1. Open a Connection to the Stereo Screen

The application must first open a connection to the stereo screen. This is done by opening screen **host:0.1**. The display pointer that is returned with the connection is then used when creating the X window that will hold the PHIGS workstation graphics.

2. Open a Window on the Stereo Screen

Again, do this by creating the X window with the stereo screen's display pointer.

3. Open a PHIGS Workstation for the Stereo Window

For example,

```
conn.drawable_id = stereo_win;
popenws(stereo_wks, (Pconnid *)(&conn),
   phigs_ws_type_x_drawable);
```

4. Set Up The Left-eye and Right-eye Views

Set up the PRP (one for each eye) in conjunction with the viewing plane. Again, the position of the viewing plane corresponds with the monitor screen. The *z*-value of the PRP is the *z*-coordinate of the PRP in VRC space. This distance should directly correspond with the distance of the user's eyes from the screen. The application should translate physical distances into distances in VRC units. Create the left-eye and right-eye viewing matrices and put one each in the view tables of the left-eye and right-eye workstations:

```
/* left eye view */
mapping.prp.x = -1.0 * half_occular_dist;
pevalviewmappingmatrix3(&mapping, &err,
                  view_rep.mapping_matrix);
psetviewrep3(stereo_wks, left_view_index, &view_rep);

/* right eye view */
mapping.prp.x = half_occular_dist;
pevalviewmappingmatrix3(&mapping, &err,
                  view_rep.mapping_matrix);
psetviewrep3(stereo_wks, right_view_index, &view_rep);
```

5. Put the GSE Stereo View Index Element Into the Structure

Put the GSE stereo view index element into the 3D structure that points to the stereo viewing matrices created in step 4.

```
popenstruct(structure);
psetviewind(stereo_view_index);
```

6. Post the Structure to the Stereo Workstation

Post the structure to the stereo workstation to view the3D object.

```
ppoststruct(stereo_wks, structure, 0.0);
```

7. Enable Screen Toggling

By default, the X server is not able to toggle between the regular screen and the stereo screen when the cursor is moved off the left- or right-side of the screen. To enable screen toggling, the application should make the following X call:

```
XScreenWarpByCursor(display,True);
```

To disable screen toggling, the application should make the following X call:

```
XScreenWarpByCursor(display,False);
```

2. Known Bugs in the System

ES/PSX Bugs

ES/PSX Bugs Fixed but Not Reported in Previous Releases

Rendering Spheres in CPK

If you run dynamic CPK with low precision spheres, rotate the spheres with their poles parallel to the monitor screen, and move the spheres a short distance, horizontal lines may flash out of the spheres. This problem has been fixed in the 1.3.1 Release.

CPK Specular Highlights

When displaying a CPK model in ES/PSX with a high precision value and the terminal emulator turned on, movement of the cursor across the screen may cause the specular highlights to blink. This problem has been fixed in the 1.3.1 Release.

CPK Spheres Clipped

In ES/PSX, spheres and lines with *z* positions that cause them to reside in the back half of the CPK window are clipped and not seen. This problem has been fixed in the 1.3.1 Release.

CPK Renderings Inverted in *z*

In ES/PSX, some CPK renderings may be inverted in *z*. This problem has been fixed in the 1.3.1 Release.

Turning Off the Display When Rendering

When a rendering is done on the PS 390, the display is turned off by sending the equivalent of a fix(1) to **TURNONDISPLAY**, which turns the wireframe display off. This prevents stenciling over the static image on the PS 390. This feature has been implemented in ES/PSX with the 1.3.1 Release.

READASCII May Crash ES/PSX

If you send a filename to **<1>readascii**, ES/PSX may crash. This problem has been fixed in the 1.3.1 Release.

DELETE STRUCTURE May Crash ES/PSX

Using the **DELETE STRUCTURE** command may crash ES/PSX. The problem occurs when a Set node in the structure that is being deleted is empty. This problem has been fixed in the 1.3.1 Release.

Set Nodes in a Structure

If you have a Set node under another Set node, which was created in **OPTIMIZE STRUCTURE** mode and which does not require the state to be saved, the X server will hang. This problem has been fixed in the 1.3.1 Release.

ES/PEX Bugs

The following ES/PEX bug has been identified in the 1.3 and 1.3.1 Releases.

4.24 EXECUTE STRUCTURE

When structures are terminated with **EXECUTE STRUCTURE** elements, problems occur when that terminating **EXECUTE STRUCTURE** element is deleted. To avoid this possibility, you should terminate with something other than an **EXECUTE STRUCTURE**.

ES/PEX Bugs Fixed in the 1.3.1 Release

The following ES/PEX bug that was reported in the *1.0 Release Notes* has been fixed in the 1.3.1 Release.

4.4 ES/PEX Polygon Rendering with ES/PSX Running

The following ES/PEX bug that was reported in the *1.3 Release Notes* has been fixed in the 1.3.1 Release.

4.15 Linetypes and Edgetypes

ES/PEX Bugs Fixed but Not Reported in Previous Releases

Back/Front Area Properties

With **FACE DISTINGUISH MODE** on and different properties set for the back and front area properties, the image will have the same properties for both. This problem has been fixed in the 1.3.1 Release.

Facet Normals

If facet normals are not specified, they are calculated from the last vertices instead of from the first vertices. This problem has been fixed in the 1.3.1 Release.

3. Documentation Corrections

ESV Workstation User's Manual

"8. Porting Guide"

1) On page 8-62, the second line of the FORTRAN example in the "Array Handling" section should read,

```
t(1,1), t(2,1), t(1,2), t(2,2), t(1,3), t(2,3)
```

2) On page 8-64, the description under the "Debugging Procedures" bulleted item should read,

"You compile programs for debugging using the **-g** option of the driver command that compiles your program and then you execute the resulting object with the **dbx** debugger."

4. Installation Instructions

In the following procedure, system output is shown in `typewriter normal` font, and user responses are shown in **`typewriter bold`** font. All user responses should be typed as shown and entered with a carriage return.

Caution: It is recommended that you be the only user logged onto the machine while loading this software.
ES/Dnet customers should note that DECnet should be stopped before the 1.3.1 Release tape is installed. Error messages will occur if this is not done. ES/Dnet can be stopped by the following command:

```
/etc/init.d/deknet stop
```

ES/Dnet can be restarted by the following command:

```
/etc/init.d/deknet start
```

To install this tape, insert the ES/Patch Tape 1.3.1 into the tape drive, login as **root**, and proceed as follows:

```
cd /
/usr/pkg/bin/inst

Software package installation

Install package relative to where [/]? <CR>
Please mount the (first, if multiple tapes)
distribution tape, then press return... <CR>
Rewinding the tape...
Verifying tape id... ok

Extracting packaging information tree... patch1.3.1

Installation Information:

Packages will be read in from the local Q24 tape drive.
Machine type: ml20

Is the information above correct? (y n) [y]? <CR>
```

```
========== checking subpackages ==========

The following subpackages may be installed:

patch-- 1.3.1 Patch Release

========== selecting subpackages ==========
```

You may select all of the above subpackages by answering "y" to the following question. If you answer "n" then you will be asked to select the optional subpackages you would like to have installed.

```
Install ALL subpackages (y n) [n]? y
========== setting system clock/calendar ==========

The current value of the clock is: Tue Dec  4 09:36:32
            MST 1990
Is the clock correct (y n) [y]? y

========== verifying single-user mode ==========

This system is not presently in a single-user run level.
Installation of a package can fail if performed at this
run level. We recommend that the system be brought to a
single user run level (using "init S") prior to
performing the installation.

Are you absolutely sure you wish to continue (y n) [n]? y

========== preserving local files ==========

No preserve list or findmods list for patch-
            preserve not executed.

========== verifying disk space ==========

Do you want to check for space (please do so unless you
really understand the consequences)  (y n) [y]? y

The system will now be checked to verify that there
is enough disk space with the current configuration
to successfully install the package (and any selected
optional subpackages). For large packages (especially
operating system packages), this can be time
consuming...
```

```
There is enough space.

========== stripping old links ==========

Stripping links for subpackage patch...

===== extracting files from subpackage archives =====

rewinding the tape...
Verifying tape id... ok
Forward spacing the tape...

Loading subpackage: patch...
Forward spacing the tape...
rewinding the tape...

========== running comply ==========
running first comply pass...
running second comply pass...
There were no comply messages from the second pass.

===== doing uncompress Tue Dec  4 09:41:32 MST 1990 ====

========== cleaning up old versions ==========

An attempt will now be made to clean up any files left
over from previous versions of the software which has
just been installed.

Searching for old versions to remove...

========== restoring preserved user files ==========

No preserve list or findmods list for patch- no files
                restored.

========== cleaning up ==========

Remove install tools (y n) [n]? y

========== installation complete ==========
```

A. PHIGS Stereo Example Program

This program can be found in **/usr/people/fstest/demo/stereo_demo.c.**

```
/*********************************************************************
 *
 *  stereo_demo.c
 *
 *      A Canonical Stereo Program to prototype stereo on ESV,
 *          release 1.3.1.
 *
 *********************************************************************/

#include <X11/Xlib.h>
#include <X11/Xatom.h>
#include <phigs.h>
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <XVideoMode.h>
#include <XInput.h>

extern int errno;

char *ProgramName;

#define     BLACK       0
#define     WHITE       1
#define     RED         2
#define     GREEN       3
#define     BLUE        4
#define     YELLOW      5
#define     CYAN        6
#define     MAGENTA     7
/* Stereo params in physical inches */
#define EYE_TO_SCREEN   36.0
#define WINDOW_SIZE     9.0
#define VIEW_RATIO      (EYE_TO_SCREEN / WINDOW_SIZE)
/* Stereo params in VRC units */
#define VRC_WINDOW_SIZE 4.0
#define INCHES_TO_VRC   (VRC_WINDOW_SIZE / WINDOW_SIZE)
#define VRC_PRP         (EYE_TO_SCREEN * INCHES_TO_VRC)
#define HALF_OCCULAR    (1.25 * INCHES_TO_VRC)
```

```
/* Dial handling */
#define XROT_DIAL        0
#define YROT_DIAL        1
#define ZROT_DIAL        2
#define SCALE_DIAL       3
#define XTRAN_DIAL       4
#define YTRAN_DIAL       5
#define ZTRAN_DIAL       6
#define DIALSCALE        330.0
#define MAXDIALS         8
#define CHARS_PER_DIAL   8
#define SPACEKEYSYM      32
#define TRUE             1
#define FALSE            0

#define FRONT_BOTTOM_LEFT       {-0.5, -0.5,  0.5}
#define FRONT_BOTTOM_RIGHT      { 0.5, -0.5,  0.5}
#define FRONT_TOP_LEFT          {-0.5,  0.5,  0.5}
#define FRONT_TOP_RIGHT         { 0.5,  0.5,  0.5}
#define BACK_BOTTOM_LEFT        {-0.5, -0.5, -0.5}
#define BACK_BOTTOM_RIGHT       { 0.5, -0.5, -0.5}
#define BACK_TOP_LEFT           {-0.5,  0.5, -0.5}
#define BACK_TOP_RIGHT          { 0.5,  0.5, -0.5}

#define FRONT       { FRONT_TOP_RIGHT,    FRONT_TOP_LEFT, \
                      FRONT_BOTTOM_LEFT,  FRONT_BOTTOM_RIGHT}
#define BACK        { BACK_TOP_LEFT,      BACK_TOP_RIGHT, \
                      BACK_BOTTOM_RIGHT,  BACK_BOTTOM_LEFT}
#define LEFT        { FRONT_TOP_LEFT,     BACK_TOP_LEFT, \
                      BACK_BOTTOM_LEFT,   FRONT_BOTTOM_LEFT}
#define RIGHT       { FRONT_TOP_RIGHT,    FRONT_BOTTOM_RIGHT, \
                      BACK_BOTTOM_RIGHT,  BACK_TOP_RIGHT}
#define TOP         { FRONT_TOP_RIGHT,    BACK_TOP_RIGHT, \
                      BACK_TOP_LEFT,      FRONT_TOP_LEFT}
#define BOTTOM      { FRONT_BOTTOM_RIGHT, FRONT_BOTTOM_LEFT, \
                      BACK_BOTTOM_LEFT,   BACK_BOTTOM_RIGHT}

/* Flags for current video state */
#define MONO     0
#define STEREO   1
int              mode_flag       = MONO;
```

```
static      Ppoint3cube[6][4] = { FRONT, BACK, LEFT, RIGHT, TOP, BOTTOM };
Pint        mono_wks= 1;
Pint        stereo_wks= 2;
Pint        structure= 1;
Pint        mono_view_index = 1;
Pint        left_view_index = 1;
Pint        right_view_index = 2;
Pfloat      priority= 1.0;
Pint        rotate_elem, translate_elem;
int         winx, winy, winw, winh;
int         mono_winx, mono_winy, mono_winw, mono_winh;
Window      mono_win, mono_rootwin, stereo_win, stereo_rootwin;
Display     *dpy, *sdpy;
GContext    gc;
GC          gc_black;
int         text_x, text_y;
float       h_to_w_aspect;
Pfloat      half_occular_dist = HALF_OCCULAR;

/* Globals for device input stuff */
XDevice          *tablet = NULL;
XID              tablet_id = 0;
XDevice          *dials = NULL;
XID              dials_id = 0;
XDeviceInfo      *devices = NULL;
int              ndevices = 0;
int              DeviceMotion = -1;
int              DevicePress = -1;
int              DeviceRelease = -1;
int              change[10];
Pmatrix3         xrotmat, yrotmat, zrotmat, scalemat, concat1,
                   concat2, concat3;
Pmatrix3         currmatrix, tranmatrix;
Pviewmapping3    mapping;
Pviewrep3        view_rep;
int              mono_fd, stereo_fd;

Window XCreateWindow();
```

```
/*********************************************************************
 *
 *  identity_matrix
 *
 *       Make an identity matrix.
 *
 *********************************************************************/

void identity_matrix( matrixit )
     Pmatrix3   matrixit;
{
  int i, j;

  for ( i = 0 ; i < 4 ; i++)
  {
      for ( j = 0 ; j < 4 ; j ++ )
      {
          if (i == j )
            matrixit[i][j] = 1.0;
          else
            matrixit[i][j] = 0.0;
      }
  }
} /* End of identity_matrix */

/*********************************************************************
 *
 *  usage
 *
 *********************************************************************/
usage ()
{
    fprintf (stderr, "usage:  %s [-options ...]\n\n", ProgramName);
    fprintf (stderr, "where options include:\n");
    fprintf (stderr, "    -display host:dpy   X server to use\n");
    fprintf (stderr, "    -geometry geom      geometry of window\n");
    fprintf (stderr, "    -b                  use backing store\n");
    fprintf (stderr, "    -fg color           set foreground color\n");
    fprintf (stderr, "    -bg color           set background color\n");
    fprintf (stderr, "    -bd color           set border color\n");
    fprintf (stderr, "    -bw width           set border width\n");
    fprintf (stderr, "\n");
    exit (1);
```

```
/*********************************************************************
 *
 *  doPhigsStuff
 *
 *********************************************************************/
doPhigsStuff(dpy, sdpy)
     Display     *dpy;
     Display     *sdpy;
{
    static Ppoint  xyzpoints[] = {{0.1, 0.1}, {0.9, 0.9}};
    Popenpexinfo   xinfo;
    Pconnid_x_drawableconn;
    Ppoint         text_pt;
    Pint       part = 1;
    Ppoint3        rep;
    Pintlst    names_to_add;

    int            i, err;
    float          deg;
    Ppoint3        tran, vrp;
    Pvector3       vpn, vup;
    Plimit         wks_viewport, wks_window;
    Pmatrix3       rotmat, comp_mat, tran1;
    Pugse0008rec   stereo_views;


    xinfo.display = dpy;
    xinfo.rdb = NULL;
    xinfo.name = NULL;
    xinfo.classname = NULL;
    xinfo.argc_p = NULL;
    xinfo.argv = NULL;
    xinfo.flags.no_monitor = 1;
    xinfo.flags.force_client_SS = 0;
    /* Open PHIGS mono */
    popenpex((Pchar*)NULL, PDEFAULT_MEM_SIZE, &xinfo);
```

```
/****** PEX DEPENDENT way of declaring the workstation type ******/

conn.display = dpy;
conn.drawable_id = stereo_win;
popenws(stereo_wks, (Pconnid *) (&conn), phigs_ws_type_x_drawable);
psethlhsrmode(stereo_wks, PHIGS_HLHSR_MODE_ZBUFF);
conn.drawable_id = mono_win;
popenws(mono_wks, (Pconnid *) (&conn), phigs_ws_type_x_drawable);
psethlhsrmode(mono_wks, PHIGS_HLHSR_MODE_ZBUFF);

wks_viewport.xmin = 0.0;
wks_viewport.xmax = winw;
wks_viewport.ymin = 0.0;
wks_viewport.ymax = winh;
psetwsviewport(stereo_wks, &wks_viewport);

wks_viewport.xmax = mono_winw;
wks_viewport.ymax = mono_winh;
psetwsviewport(mono_wks, &wks_viewport);

wks_window.xmin = 0.0;
wks_window.xmax = 1.0;
wks_window.ymin = 0.0;
wks_window.ymax = 1.0;
psetwswindow(stereo_wks, &wks_window);
psetwswindow(mono_wks, &wks_window);

/* Set Background color; entry 0 of table. */
rep.x = 0.50;
rep.y = 0.50;
rep.z = 0.50;
psetcolourrep(stereo_wks, 0, &rep );
psetcolourrep(mono_wks, 0, &rep );

rep.x = 1.0;
rep.y = 0.0;
rep.z = 0.0;
psetcolourrep(stereo_wks, RED, &rep );
psetcolourrep(mono_wks, RED, &rep );
```

```
rep.x = 0.0;
rep.y = 1.0;
rep.z = 0.0;
psetcolourrep(stereo_wks, GREEN, &rep );
psetcolourrep(mono_wks, GREEN, &rep );

rep.x = 0.0;
rep.y = 0.0;
rep.z = 1.0;
psetcolourrep(stereo_wks, BLUE, &rep );
psetcolourrep(mono_wks, BLUE, &rep );

rep.x = 1.0;
rep.y = 1.0;
rep.z = 0.0;
psetcolourrep(stereo_wks, YELLOW, &rep );
psetcolourrep(mono_wks, YELLOW, &rep );

rep.x = 0.0;
rep.y = 1.0;
rep.z = 1.0;
psetcolourrep(stereo_wks, CYAN, &rep );
psetcolourrep(mono_wks, CYAN, &rep );

rep.x = 1.0;
rep.y = 0.0;
rep.z = 1.0;
psetcolourrep(stereo_wks, MAGENTA, &rep );
psetcolourrep(mono_wks, MAGENTA, &rep );

/* Viewing Stuff Common to Stereo and Mono */
/* Set up the left and right eye views */
vrp.x = 0.0;    vrp.y = 0.0;    vrp.z = 0.0;
vpn.x = 0.0;    vpn.y = 0.0;    vpn.z = 1.0;
vup.x = 0.0;    vup.y = 1.0;    vup.z = 0.0;
pevalvieworientationmatrix3(&vrp, &vpn, &vup, &err,
                            view_rep.orientation_matrix);
view_rep.clip_limit.xmin = 0.0;
view_rep.clip_limit.xmax = 1.0;
view_rep.clip_limit.ymin = 0.0;
view_rep.clip_limit.ymax = 1.0;
view_rep.clip_limit.zmin = 0.0;
view_rep.clip_limit.zmax = 1.0;
view_rep.clip_xy = PCLIP;
```

```
view_rep.clip_back = PNOCLIP;
view_rep.clip_front = PNOCLIP;

mapping.window.xmin = -1.0 * VRC_WINDOW_SIZE/2.0;
mapping.window.xmax =  VRC_WINDOW_SIZE/2.0;
mapping.window.ymin = -1.0 * VRC_WINDOW_SIZE/2.0;
mapping.window.ymax =  VRC_WINDOW_SIZE/2.0;
mapping.viewport.zmin = 0.0;
mapping.viewport.zmax = 1.0;
mapping.proj = PPERSPECTIVE;
mapping.prp.y = 0.0;
mapping.prp.z = VRC_PRP;
mapping.view_plane = -0.0;
mapping.back_plane = -100.0;
mapping.front_plane = mapping.prp.z - 1.0;

/* Viewing Stuff for Mono eye view */
mapping.viewport.xmin = 0.0;
mapping.viewport.xmax = 1.0;
mapping.viewport.ymin = 0.0;
mapping.viewport.ymax = 1.0;
mapping.prp.x = 0.0;
pevalviewmappingmatrix3(&mapping, &err,
                        view_rep.mapping_matrix);
psetviewrep3(mono_wks, mono_view_index, &view_rep);

/* Viewing Stuff for Stereo views */
mapping.viewport.xmin = 0.0;
mapping.viewport.xmax = 1.0;
mapping.viewport.ymin = 0.0;
mapping.viewport.ymax = 1.0;

/* left eye view */
mapping.prp.x = -1.0 * half_occular_dist;
pevalviewmappingmatrix3(&mapping, &err,
                        view_rep.mapping_matrix);
psetviewrep3(stereo_wks, left_view_index, &view_rep);

/* right eye view */
mapping.prp.x = half_occular_dist;
pevalviewmappingmatrix3(&mapping, &err,
                        view_rep.mapping_matrix);
psetviewrep3(stereo_wks, right_view_index, &view_rep);
```

```
/* Build the geometry structure */
/* ---------------------------- */
popenstruct( structure);

/* psetviewind(mono_view_index); */
stereo_views.mono = left_view_index;
stereo_views.left = left_view_index;
stereo_views.right = right_view_index;
pgse(PUGSE_STEREO_VIEW_INDICES, &stereo_views);
psethlhsrid(PHIGS_HLHSR_ID_ON);

/* Translate node for dials */
tran.x = 0.0;
tran.y = 0.0;
tran.z = 0.0;
translate_elem = 3;
ptranslate3( &tran, &err, tran1 );
psetlocaltran3( tran1, PPRECONCATENATE );

/* set initial degree of rotation to 0 */
deg = 0.0;
rotate_elem = 4;
protatez( deg, &err, comp_mat );
psetlocaltran3( comp_mat, PPRECONCATENATE );

/* Translate node to offset object a little */
tran.x = 0.50;
tran.y = 0.0;
tran.z = 0.0;
ptranslate3( &tran, &err, tran1 );
psetlocaltran3( tran1, PPRECONCATENATE );

psetintstyle( PSOLID);

psetfacecullmode( PCULL_NONE );
psetfacedistgmode( PDISTG_NO );

psetintcolourind( BLUE);/* blue */
pfillarea3( 4, cube[0]);

psetintcolourind( GREEN);/* green */
pfillarea3( 4, cube[1]);
```

```
    psetintcolourind( RED);      /* red */
    pfillarea3( 4, cube[2]);

    psetintcolourind( YELLOW);   /* yellow */
    pfillarea3( 4, cube[3]);

    psetintcolourind( CYAN);     /* cyan */
    pfillarea3( 4, cube[4]);

    psetintcolourind( MAGENTA); /* magenta */
    pfillarea3( 4, cube[5]);

    pclosestruct();

    /* set edit mode to replace */
    pseteditmode(PEDIT_REPLACE);
    /* set display updates to wait */
    psetdisplayupdatest(stereo_wks, PWAIT, PNIVE);
    psetdisplayupdatest(mono_wks, PWAIT, PNIVE);
    /* post the structure */
    ppoststruct(stereo_wks, structure, 0.0);
    ppoststruct(mono_wks, structure, 0.0);
    /* Do the first display */
    pupdatews( mono_wks, PPERFORM );

    /* Go look for dial events, etc. */
    MyMainLoop();

    pclosews(stereo_wks);
    pclosews(mono_wks);
    pclosephigs();

} /* End of doPhigsStuff */
```

```
/*****************************************************************
 *
 *  MyMainLoop
 *
 *     Custom loop so we can catch events
 *     associated with the extensions to X Input and apply them to
 *     PHIGS structures.
 *
 *****************************************************************/

int MyMainLoop()

{
    int                 i, dials_values[10],error,events_waiting,nfds;
    unsigned long       readfds, writefds, exceptfds;
    unsigned long       monofdmask, stereofdmask;
    static int          rotate_dirty, tran_dirty;
    Pvector3            tranval, scaleval;
    XEvent              event, peek_event;
    XAnyEvent           *any_event;
    XButtonEvent        button_event;
    XDeviceMotionEvent  *dm;
    Display             *current_dpy;

    /* Set up file descriptor masks */
    monofdmask = 1 << mono_fd;
    stereofdmask = 1 << stereo_fd;

    /* Initialize the current screen pointer. */
    current_dpy = dpy;

    /* Initialize the rotation-scale accumulation matrix */
    identity_matrix(currmatrix);

    /* Initialize the translation accumulation vector */
    tranval.x = 0.0; tranval.y = 0.0; tranval.z = 0.0;
    /* Initialize the scale accumulation vector */
    scaleval.x = 1.0; scaleval.y = 1.0; scaleval.z = 1.0;

    /* Do forever */
    for (;;)
    {
            /* Use "select" to sleep on input from the two screens */
            readfds = monofdmask | stereofdmask;
```

```
writefds = 0; exceptfds = 0;
nfds = select(32, &readfds, &writefds, &exceptfds, NULL);
if (nfds == 0)
    continue;

/* First get the event */
/* Check the Mono comm line */
if ( (readfds & monofdmask) != 0)
{
    if ( (events_waiting = XPending(dpy)) != 0)
    {
      current_dpy = dpy;
      XNextEvent(current_dpy, &event);
      any_event = (XAnyEvent *) &event;
      mode_flag = MONO;
    }
}
/* Check the Stereo comm line */
else if ( (readfds & stereofdmask) != 0)
{
    if ( (events_waiting = XPending(sdpy)) != 0)
    {
      current_dpy = sdpy;
      XNextEvent(current_dpy, &event);
      any_event = (XAnyEvent *) &event;
      mode_flag = STEREO;
    }
}
else
{
    continue;
}

if ( (event.type == MapNotify)    ||
     (event.type == Expose)       ||
     (event.type == ConfigureNotify) )
{
    if (any_event->window == mono_win)
    {
      predrawallstruct(mono_wks, PALWAYS);
      XSync(current_dpy, 0);
    }
    else if (any_event->window == stereo_win)
    {
```

```
        predrawallstruct(stereo_wks, PALWAYS);
        XSync(current_dpy, 0);
      }
  }
  else if (event.type == LeaveNotify)
  {
      if (any_event->window == stereo_rootwin)
      {
        current_dpy = dpy;
      }
      else if (any_event->window == mono_rootwin)
      {
        current_dpy = sdpy;
      }

  }

  else if (event.type == DeviceMotion)
    {
      do
      {
        peek_event.type = 0;
        /* If this is a tablet event */
        dm = (XDeviceMotionEvent *) &event;
        if (dm->deviceid == tablet_id)
        {
            printf("Tablet Motion\n");
        }
          /* If this is a dial event */
        else if (dm->deviceid == dials_id)
        {
            /* printf("Dial event.\n"); */
            /* Now process the data for all dials. */
            for (i = 0; i < dm->axes_count; i++)
            {
              /* Copy the new dial value */
              if (dm->axis_data[i] != 0)
              {
                  dials_values[i + dm->first_axis ] =
                    dm->axis_data[i];
                  /* Set the dirty flag */
                  change[i+dm->first_axis] = 1;
              }
            }
```

```
        if (XPending(current_dpy) > 0)
        {
          XPeekEvent( current_dpy, &peek_event );
          if ( peek_event.type == DeviceMotion )
              XNextEvent(current_dpy, &event);
        }

    } /* End of if this is a dial event */

} while ( peek_event.type == DeviceMotion );

/* Now Edit the PHIGS structure */

/* Do the rotation matrices */
rotate_dirty = FALSE;
if (change[XROT_DIAL])
{
  protatex( (float)dials_values[XROT_DIAL]/DIALSCALE,
          &error, xrotmat );
  rotate_dirty = TRUE;
}
else
  identity_matrix( xrotmat );


if (change[YROT_DIAL])
{
  protatey( (float)dials_values[YROT_DIAL]/DIALSCALE,
          &error, yrotmat );
  rotate_dirty = TRUE;
}
else
  identity_matrix( yrotmat );

if (change[ZROT_DIAL])
{
  protatez( (float)dials_values[ZROT_DIAL]/DIALSCALE,
          &error, zrotmat );
  rotate_dirty = TRUE;
}
else
  identity_matrix( zrotmat );

if (change[SCALE_DIAL])
```

```
{
  scaleval.x = 1.0 +
      (float)dials_values[SCALE_DIAL]/(10.0*DIALSCALE);
  scaleval.y = scaleval.x;
  scaleval.z = scaleval.x;
  pscale3( &scaleval, &error, scalemat );
  rotate_dirty = TRUE;
}
else
  identity_matrix( scalemat );

if (rotate_dirty == TRUE)
{
  pcomposematrix3( currmatrix, xrotmat, &error, concat1 );
  pcomposematrix3( concat1, yrotmat, &error, concat2 );
  pcomposematrix3( concat2, zrotmat, &error, concat3 );
  pcomposematrix3(concat3, scalemat, &error, currmatrix );
  popenstruct(structure);
  psetediteditmode(PEDIT_REPLACE);
  psetelemptr(rotate_elem);
  psetlocaltran3( currmatrix,PPRECONCATENATE );
  pclosestruct();
}

/* Change Translation Matrix */
tran_dirty = FALSE;
if (change[XTRAN_DIAL])
{
  tranval.x += (float)dials_values[XTRAN_DIAL]/DIALSCALE;
  tran_dirty = TRUE;
}
if (change[YTRAN_DIAL])
{
  tranval.y +=(float)dials_values[YTRAN_DIAL]/DIALSCALE;
  tran_dirty = TRUE;
}

if (change[ZTRAN_DIAL])
{
  tranval.z += (float)dials_values[ZTRAN_DIAL]/DIALSCALE;
  tran_dirty = TRUE;
}
```

```
                if (tran_dirty == TRUE)
                {
                  ptranslate3(&tranval, &error, tranmatrix);
                  popenstruct(structure);
                  pseteditmode(PEDIT_REPLACE);
                  psetelemptr(translate_elem);
                  psetlocaltran3( tranmatrix,PPRECONCATENATE );
                  pclosestruct();
                }


                if (mode_flag == STEREO)
                {
                  pupdatews(stereo_wks, PPERFORM);
                }
                else
                {
                  pupdatews(mono_wks, PPERFORM);
                }

                for (i=0; i<8; i++)
                  change[i] = 0;

            } /* End of if event is DeviceMotion */

    } /* End of do forever */

} /* End of MyMainLoop */
```

```
/*****************************************************************
 *
 *  main
 *
 *****************************************************************/
main(argc, argv)
     int    argc;
     char   *argv[];
{
    int             amount, i, j ;

    char            *geom = NULL;
    register int    xdir, ydir;
    register int    xoff, yoff;
    register int    centerX, centerY;
    XSizeHints      hints;
    XGCValues       xgcv;
    XSetWindowAttributes xswa;
    XWindowAttributes    winattr;
    Window          root;
    Colormap        map;
    int             x, y, w, h;
    int             mono_button_x, mono_button_y,
                      mono_button_w, mono_button_h;
    int             mono_button_bw, mono_button_bc, mono_button_bckgrnd;
    Font            font;
    char            *fontname;
    XFontStruct     *font_info;
    XGCValues       gcvals;
    unsigned        valmask;
    unsigned int    d;
    unsigned int    bw = 1;
    char            *display = NULL;
    char            *stereoDisplayString;
    Status          status;
    char            *strptr;
    char            *fg = NULL;
    char            *bg = NULL;
    char            *bd = NULL;
    int             fg_pix, bg_pix, bd_pix;
    XColor          fg_def, fg_exact, bg_def, bg_exact, bd_def, bd_exact;
    int             bs = NotUseful;
    Visual          visual;
```

```
ProgramName = argv[0];

for (i=1; i < argc; i++)
{
        char *arg = argv[i];

        if (arg[0] == '-') {
            switch (arg[1]) {
              case 'd':   /* -display host:dpy */
                  if (++i >= argc) usage ();
                  display = argv[i];
                  continue;
              case 'g':   /* -geometry host:dpy */
                  if (++i >= argc) usage ();
                  geom = argv[i];
                  continue;
              case 'b':   /* -b or -bg or -bd */
                  if (!strcmp(argv[i], "-bg")) {
                    if (++i >= argc) usage ();
                    bg = argv[i];
                  } else if (!strcmp(argv[i], "-bd")) {
                    if (++i >= argc) usage ();
                    bd = argv[i];
                  } else if (!strcmp(argv[i], "-bw")) {
                    if (++i >= argc) usage ();
                    bw = atoi(argv[i]);
                  } else
                    bs = Always;
                  continue;
              case 'f':   /* assume -fg */
                  if (++i >= argc) usage ();
                  fg = argv[i];
                  continue;
              default:
                  usage ();
            }
        } else if (argv [i] [0] == '=') /* obsolete */
                geom = argv[i];
        else
            usage ();
}
```

```
if (!(dpy = XOpenDisplay(display)))
{
        perror("Cannot open display\n");
        exit(-1);
}

mono_fd = XConnectionNumber(dpy);

/* Open the stereo screen for stereo windows. */
stereoDisplayString = XDisplayString(dpy);
strptr = rindex(stereoDisplayString, ':');
sprintf( (strptr + 1), "0.1");
if (!(sdpy = XOpenDisplay(stereoDisplayString) ))
{
        perror("Cannot open display for Stereo screen\n");
        exit(-1);
}
stereo_fd = XConnectionNumber(sdpy);
if (fg) {
        status = XAllocNamedColor(dpy, map, fg, &fg_def, &fg_exact);
        fg_pix = status ? fg_def.pixel :
             WhitePixel(dpy, DefaultScreen(dpy));
} else
        fg_pix = WhitePixel(dpy, DefaultScreen(dpy));

if (bg) {
        status = XAllocNamedColor(dpy, map, bg, &bg_def, &bg_exact);
        bg_pix = status ? bg_def.pixel :
            BlackPixel(dpy, DefaultScreen(dpy));
} else
        bg_pix = BlackPixel(dpy, DefaultScreen(dpy));

if (bd) {
        status = XAllocNamedColor(dpy, map, bd, &bd_def, &bd_exact);
        bd_pix = status ? bd_def.pixel :
            WhitePixel(dpy, DefaultScreen(dpy));
} else
        bd_pix = WhitePixel(dpy, DefaultScreen(dpy));
```

```
if (geom)
{
    (void) XParseGeometry(geom, &mono_winx, &mono_winy,
                          &mono_winw, &mono_winh);
}


xswa.backing_store = bs;
xswa.event_mask = ExposureMask | StructureNotifyMask;
xswa.background_pixel = bg_pix;
xswa.border_pixel = bd_pix;
visual.visualid = CopyFromParent;

/* Select leave events on the mono root window */
mono_rootwin = RootWindow(dpy, DefaultScreen(dpy));
XSelectInput(dpy, mono_rootwin, LeaveWindowMask);

/* Create Mono window */
if (mono_winh == 0)
        mono_winh = 500;
if (mono_winw == 0)
        mono_winw = 500;
mono_win = XCreateWindow(dpy,
            RootWindow(dpy, DefaultScreen(dpy)),
            mono_winx, mono_winy, mono_winw, mono_winh, bw,
            DefaultDepth(dpy, DefaultScreen(dpy)), InputOutput,
            &visual,
            CWEventMask | CWBackingStore | CWBorderPixel | CWBackPixel,
            &xswa);
XChangeProperty(dpy, mono_win, XA_WM_NAME, XA_STRING, 8,
                PropModeReplace, "Stereo Demo", sizeof("Stereo Demo") );
XMapWindow(dpy, mono_win);
/* Open input devices, select on dial events */
SensitizeWindow(dpy, mono_win);
XSync(dpy, 0);


/* Select leave events on the stereo root window */
stereo_rootwin = RootWindow(sdpy, DefaultScreen(sdpy));
XSelectInput(sdpy, stereo_rootwin, LeaveWindowMask);
```

```
    /* Open the stereo window on the Stereo Screen */
    winx = 0;
    winy = 0;
    winh = (XDisplayHeight(sdpy, 1)) - 100;
    winw = winh;
    stereo_win = XCreateWindow(sdpy,
                RootWindow(sdpy, DefaultScreen(sdpy)),
                winx, winy, winw, winh, bw,
                DefaultDepth(sdpy, DefaultScreen(sdpy)), InputOutput,
                &visual,
                CWEventMask | CWBackingStore | CWBorderPixel | CWBackPixel,
                &xswa);
    XSync(sdpy, 0);
    XChangeProperty(sdpy, stereo_win, XA_WM_NAME, XA_STRING, 8,
                PropModeReplace, "Stereo Demo", 11);
    hints.flags = USPosition | USSize ;
    hints.x     = winx;
    hints.y     = winy;
    hints.width = winw;
    hints.height= winh;
    XSetNormalHints(sdpy, stereo_win, &hints);
    XSetTransientForHint(sdpy, RootWindow(sdpy, DefaultScreen(sdpy)),
                         stereo_win );
    XSync(sdpy, 0);
    /* Open input devices, select on dial events */
    SensitizeWindow(sdpy, stereo_win);
    XMapWindow(sdpy, stereo_win);
    XFlush(sdpy, 0);

    doPhigsStuff(dpy, sdpy);

}   /* End of main */
```

```
/*****************************************************************
 *
 *  SensitizeWindow
 *
 *     Routine to select input events for PHIGS graphics windows.
 *
 *****************************************************************/

int SensitizeWindow(my_dpy, window)
     Display     *my_dpy;
     Window      window;

{
    int           i, j;
    int           eventCount = 0;
    XEventClass   eventClass[100];
    XStringFeedbackControl strfc;
    KeySym        ledstring[MAXDIALS][CHARS_PER_DIAL],
                      blankled[CHARS_PER_DIAL];
    static char   textstring[MAXDIALS][CHARS_PER_DIAL] =
                   { " X ROT  ", " Y ROT  ", " Z ROT  ", " SCALE  ",
                         " X TRAN ", " Y TRAN ", " Z TRAN ", "        " };

    /* Get a list of the available devices */
    devices = XListInputDevices(my_dpy, &ndevices);

    /* Open the input devices */
    for (i = 0; i < ndevices; i++, devices++)
    {
        if (strcmp("TABLET", devices->name) == 0)
        {
            tablet_id = devices->id;
            tablet = XOpenDevice(my_dpy, tablet_id);
        }

        if (strcmp("KNOB_BOX", devices->name) == 0)
        {
            dials_id = devices->id;
            dials = XOpenDevice(my_dpy, dials_id);
        }
    }

    eventCount = 0;
```

```
if (!tablet)
{
    fprintf(stderr, "No TABLET\n");
}
else
{
    DeviceMotionNotify(tablet, DeviceMotion,
                       eventClass[eventCount]);
    eventCount++;

    DeviceButtonPress(tablet, DevicePress, eventClass[eventCount]);
    eventCount++;

    DeviceButtonRelease(tablet, DeviceRelease,
                        eventClass[eventCount]);
    eventCount++;
}
if (!dials)
{
    fprintf(stderr, "No DIALS\n");
}
else
{
    DeviceMotionNotify(dials, DeviceMotion, eventClass[eventCount]);
    eventCount++;
}


if (!dials && !tablet)
{
    exit(0);
}

XFreeDeviceList(devices);

XSelectExtensionEvent(my_dpy, window, eventClass, eventCount);
```

```
    /* Set the dial labels */
    /* Load the keysym arrays */
    for (i = 0; i < MAXDIALS; i++)
      for (j = 0; j < CHARS_PER_DIAL; j++)
      {
            ledstring[i][j] = (KeySym) textstring[i][j];/* Dial labels */
            blankled[j] = SPACEKEYSYM;                  /* Blank labels */
      }

    strfc.class = StringFeedbackClass;
    strfc.length = sizeof(XStringFeedbackControl);
    strfc.num_keysyms = CHARS_PER_DIAL;
    for (i=0; i<MAXDIALS; i++)
    {
            strfc.id = i;
            strfc.syms_to_display = ledstring[i];
            XChangeFeedbackControl(my_dpy, dials, DvString, &strfc);
    }

    return;

} /* End of SensitizeWindow */


/* End of stereo_demo.c */
```