Part Number: 800-1256-04
Version: C of 17 March 1986
For: Sun System Release 2.3

SunCGI Reference Manual

Acknowledgements

Contents

Tables

Figures

Preface

This document describes **SunCGI**, an implementation of the ANSI *Computer Graphics Interface* (CGI) by Sun Microsystems, Inc. Previously, CGI was known as the *Virtual Device Interface* (VDI) standard. Appendix B summarizes the differences between **SunCGI** and ANSI CGI.

The CGI standard is currently under development. Future releases of SunCGI will reflect changes in ANSI CGI.

Controlling Document

The following document was used in interpreting the CGI standard:

[1] ANSI X3H3 84/85. *Information Processing Computer Graphics Virtual Device Interface (VDI) Functional Description*. March 1984.

Audience

The intended reader of this document is an applications programmer who is familiar with interactive computer graphics and the C programming language. This manual contains several example programs that can be used as templates for larger **SunCGI** applications.

Documentation Conventions

Italic font is used to indicate file names, function arguments, variables and internal states of **SunCGI**. Italics are also used in the conventional manner (to emphasize important words and phrases). ALL CAPS is used to indicate values in enumerated types. **Bold font** is used for the names of Sun software packages. Function names are printed with `constant width font`.

1

Introduction

1

Introduction

SunCGI provides access to low-level graphics device functions without the restrictions, benefits, or overhead of higher-level graphics packages like SunCore. SunCGI is useful for 2D graphics programs which do not require segmentation or transformations. The absence of segmentation from SunCGI makes drawing diagrams faster and simpler, but does not provide automatic picture regeneration. SunCGI programs are usually smaller and more efficient than SunCore programs with similar functionality. In addition, SunCGI programs will run on Sun devices without explicitly specifying the device at compile time. SunCGI provides output primitives (for example, circles), attributes (for example, sophisticated pattern filling), and input primitives which are not offered by SunCore. The CGI standard is currently under development, and therefore, CGI has not been accepted by the X3H3 committee, ANSI, or the computer graphics community. Only certain models within CGI are supported by SunCGI. Specifically SunCGI implements input option sets 1, 2, 3, 4, and 6 and output option sets 1 through 6 of the CGI standard. CGI does not support 3D output primitives.

SunCGI *does* provides output primitives, attribute selection, and input device management, at a level which is close to the actual device driver; thus affording speed and flexibility not offered by higher-level graphics packages like SunCore. SunCGI provides output primitives which are not provided by any of the other Sun graphics packages: for example disjoint polygons, circles, ellipses, and cell arrays (which can be thought of as scaled and transformed pixel arrays). CGI also provides a larger vocabulary of attributes than SunCore. SunCGI also provides facilities for explicitly binding virtual input devices to physical input devices as well as explicit management of an *event queue*.

1.1. Using SunCGI

Here is a SunCGI example application program written in C:

```
#include <cgidefs.h>

Ccoor box[5] = { 10000,10000 ,
                 10000,20000 ,
                 20000,20000 ,
                 20000,10000 ,
                 10000,10000 };

main()
{
    Ccoorlist boxlist;
    Cint name;
    Cvwsurf device;

    boxlist.n = 5;
    boxlist.ptlist = box;
    NORMAL_VWSURF(device, PIXWINDD);

    open_cgi();
    open_vws(&name, &device);

    polyline(&boxlist);
    sleep(10);

    close_vws(name);
    close_cgi();
}
```

Figure 1-1 *Simple Example Program*

SunCGI uses a variety of structures and enumerated types shown in Appendix C. The file `<cgidefs.h>` should be included in each SunCGI application program to provide necessary definitions and constants.

Here is an example of a command line for compiling `box.c` to run in the SunView environment:

```
% cc box.c -o box -lcgi -lsunwindow -lpixrect -lm
```

The order in which the libraries are linked to the program is important.

All SunCGI functions can be called by one of two names: the expanded name (default) or the C language binding name. See Appendix H for information on the list of names for the shorter C language binding.

As a final note, do not name any user-defined function or variable starting with the letters `_cgi` because doing so may disrupt the internal workings of SunCGI.

FORTRAN programmers can access SunCGI functions by using the include file in `cgidefs77.h` and using the `/usr/lib/libcgi77.a` library to link with. Details of the FORTRAN interface to SunCGI are provided in Appendix G.

1.2. The SunCGI Lint Library

SunCGI provides a *lint* library which provides type checking beyond the capabilities of the C compiler. For example, you could use the SunCGI *lint* library to check a program called `glass.c` with command like this:

```
% lint glass.c -lcgi
```

Note that the error messages that *lint* generates are mostly warnings, and may not necessarily have any effect on the operation of the program. For a detailed explanation of `lint`, see the *lint* chapter in the *Programming Tools* manual.

1.3. Overview of SunCGI

This section provides an overview of the substance of this manual. The four major sections of the manual (which correspond to chapters) are:

1) view surface initialization and termination (control),

2) output primitives,

3) attributes, and

4) input.

The overview of these chapters contains a brief introduction to the basic concepts of CGI. The appendices at the end of this manual provide quick reference tables and descriptions of the interfaces between SunCGI and

1) SunView and

2) FORTRAN.

Initialization and Termination

Chapter 2 describes functions for

1) initializing and terminating the entire SunCGI package and individual view surfaces,

2) defining the coordinate systems,

3) interface negotiation, and

4) signal trapping.

The first section Chapter 2 describes functions for opening and closing view surfaces (which are either windows or screens). SunCGI provides facilities for writing primitives to multiple view surfaces. Output primitives can be written to a selected subset of the open view surfaces by using the `activate_vws` and `deactivate_vws` functions (which turn a view surface on or off without closing the view surface or affecting the display). The functions discussed in Chapter 2 also define the range of virtual device coordinates (VDC space) and device coordinates (screen space). The coordinates of most SunCGI functions are expressed in terms of VDC space. The limits of both VDC space and screen space can be defined by the application program.

If you are attempting to run an application program developed on another vendor's version of CGI, negotiation functions are provided which describe the capabilities of SunCGI. The application program can use the information obtained by using the negotiation functions to call appropriate functions in

SunCGI to make the application program run correctly. Finally, Chapter 2 describes SunCGI's option for trapping SIGWINCH signals (generated by manipulating the window environment which the application program is using).

Output Primitives

SunCGI provides functions for drawing geometrical output primitives (for example, polygons, circles, and ellipses) as well as functions for performing raster operations. The coordinates of output primitives are specified in VDC space (with the exception of some raster functions). Geometrical output primitives include rectangles, polymarkers, circular and elliptical arcs. Geometrical output primitives are affected by attributes described in Chapter 4 (like fill style and line width). All output primitives are affected by the *drawing mode* which determines how an output primitives is affected by pixels which have been previously drawn on the screen.

Attributes

Attribute functions control the appearance of output primitives. Attributes can be set individually, or in groups which are called bundles. The use of most attributes is fairly straightforward; fill textures require a word of explanation. Geometrical output primitives can be filled with textures called hatches or patterns. Hatches are simply arrays of color values with each element of the array corresponding to a pixel. Patterns are arrays of color values which can be scaled and translated.

Input

SunCGI offers a standard interface for receiving input from the mouse and the keyboard. The CGI input model is based on the logical input device model in GKS. In this system, a logical input device (for example, a LOCATOR device), is bound to a physical device (for example, the *x-y* position of the mouse) called a trigger. Triggers may be associated with logical input devices by the application program. Each logical input device has an associated measure (for example, the measure of a LOCATOR device is the mouse position on the screen). Each logical input device also has a state which determines how a device handles input. Each logical input device can be in one of five states:

1) RELEASED (uninitialized),
2) NO_EVENTS (initialized but unable to receive input),
3) REQUEST_EVENT (waiting for one event),
4) RESPOND_EVENT (report one event asynchronously), and
5) QUEUE_EVENT (put each event at the end of the *event queue*).

Errors

Errors are reported in SunCGI by setting the return value of the function to a nonzero result and echoing an error message and number on the terminal. However, error trapping can be controlled by the `set_error_warning_mask` function. An explanation of each error message (and suggestions for how to eliminate them) is presented in Appendix D.

Programming Tips

For novice C language users, the syntax of **SunCGI** may pose some initial difficulties. When a pointer is specified as an argument to a **SunCGI** function, **SunCGI** usually expects space to be allocated by the application program and the function argument to be preceded by an ampersand (&). **SunCGI** uses many enumerated types. These types are printed by the `printf` function as integers. If you want to print out these values in English, you should use the enumerated types as indices into a character array which contains appropriate English equivalents of the enumerated types. Finally, if you are a novice programmer, copy the example programs in Appendix E and use them as templates to build your own program with. Further help can be obtained by referring to the tables at the end of Appendix D. These tables list commonly encountered problems and how to solve them.

Appendices

The first five appendices are intended to make **SunCGI** easier to understand. This information will probably be particularly useful to novice users. The last two appendices describe the interfaces:

1. between **SunCGI** and **SunView**, and
2. between **SunCGI** and the FORTRAN programming language.

Appendix A explains the difference between **SunCGI** and **SunCore**. Appendix B lists the ANSI CGI standard functions which are not implemented by **SunCGI** and the **SunCGI** functions which are not part of the ANSI CGI standard. Appendix C provides the type definitions used by the **SunCGI** functions. Appendix D lists the error messages and possible strategies for eliminating them. Appendix D also lists possible causes of simple run-time errors. Appendix E describes sample programs.

The final two appendices describe the interfaces between **SunCGI** and other Sun software packages: **SunView** and FORTRAN. The first of the two interface appendices explains how to call **SunCGI** from application programs written on top of **SunView**. This interface allows **SunCGI** to write output primitives in different windows using different attributes. This interface is useful for application programs which wish to control different areas of the view surface independently. Appendix G describes the interface to the FORTRAN programming language. The behavior of each **SunCGI** function is the same in both C and FORTRAN.

1.4. References

[1] ANSI X3H3. *Computer Graphics Virtual Device Interface*. March 1984.

[2] J.D. Foley and A. van Dam. *Fundamentals of Interactive Computer Graphics*. Addison-Wesley, 1982.

[3] B.W. Kernighan and D.M. Ritchie. *The C Programming Language*. Prentice-Hall, 1978.

[4] W.M. Newman and R.F. Sproull. *Principles of Interactive Computer Graphics*. McGraw-Hill, 1979.

[5] V.R. Pratt. *Standards and Performance Issues in the Workstation Market*. IEEE Computer Graphics and Applications, April 1984.

[6] *SunView Programmer's Guide*. Sun Microsystems.

[7] *SunView System Programmer's Guide*. Sun Microsystems.

[8] *Pixrect Reference Manual*. Sun Microsystems.

[9] *SunCore Reference Manual*. Sun Microsystems.

2

Initializing and Terminating SunCGI

2

Initializing and Terminating SunCGI

The current CGI standard does not provide functions for initializing and terminating devices. ANSI CGI is intended to provide an interface for a single view surface (one per CGI instance). **SunCGI** extends CGI into the window environment by allowing a single CGI process to control multiple view surfaces. Six nonstandard functions `open_cgi`, `close_cgi`, `open_vws`, `close_vws`, `activate_vws`, and `deactivate_vws` are included in **SunCGI**. `open_cgi` and `close_cgi` initialize and terminate the operation of the **SunCGI** package. A view surface is initialized and terminated with `open_vws` and `close_vws`. A view surface is automatically activated when it is opened. **SunCGI** is capable of handling more than one view surface at once. Output primatives can be restricted from a view surface with `deactivate_vws`.

2.1. View Surface Initialization and Selection

A view surface is automatically activated when it is opened. However, a view surface can be deactivated (with the `deactivate_vws` function) when the output stream is not intended to appear on all view surfaces. Subsequent calls to **SunCGI** output functions will not apply to deactivated view surfaces[1] until `activate_vws` is called again (see the following example).

[1] However, inputs can be received on deactivated view surfaces.

```
#include <cgidefs.h>

main()
{
    Ccoor bot, top, center;
    Cint name1, name2, radius;
    Cvwsurf device1, device2;

    bot.x = 5000;
    bot.y = 5000;
    center.x = 10000; -
    center.y = 10000;
    radius = 5000;
    top.x = 15000;
    top.y = 15000;

    open_cgi();
    NORMAL_VWSURF(device1, PIXWINDD);
    open_vws(&name1, &device1);
    NORMAL_VWSURF(device2, PIXWINDD);
    open_vws(&name2, &device2);

    rectangle(&bot, &top);
    deactivate_vws(name2);
    circle(&center, radius);
    activate_vws(name2);
    circle(&center, 2*radius);

    sleep(20);

    close_vws(name1);
    close_vws(name2);
    close_cgi();
}
```

Figure 2-1 *Example Program with Multiple Workstations*

Open CGI (SunCGI Extension)

`Cerror open_cgi()`

`open_cgi` initializes the state of SunCGI to CGOP (CGi OPen). `open_cgi` does not initialize input devices but does initialize the *event queue*. No other CGI functions can be used without generating an error if `open_cgi` has not been called. SunCGI traps various signals as described in Section 2.3.

Errors

`ENOTCGCL` [1] CGI not in proper state: CGI shall be in state CGCL.

Table 2-1 *SunCGI Default States*

State	*Value*
Range of VDC space	0-32767 in both *x* and *y* directions
Clip Indicator	CLIP
Clip Rectangle	Range of VDC space
Error Warning Mask	INTERRUPT
Input Devices	Uninitialized
Input Queue	EMPTY
Trigger Associations	Defaults specific values listed in Table 5-4
Echo Modes	Device specific values listed in Table 5-5

You may be unfamiliar with some of the entries discussed in Table 2-1. However, these concepts are explained in the course of this chapter. Further, each of these concepts are referenced in the index.

Open View Surface (SunCGI Extension)

```
Cerror open_vws(name, devdd)
Cint *name; /* name assigned to cgi view surface */
Cvwsurf *devdd; /* view surface descriptor */
```

`open_vws` initializes a view surface. The list of available view surfaces is described below in Table 2-2. `open_vws` initializes the attributes to their default values (listed in Table 2-3). The returned argument *name* is the identifier which is used to refer this view surface in other **SunCGI** functions. To reinitialize the state of the view surface without reopening it, use the `hard_reset` function.

More than one view surface can be open at one time. Output primitives are displayed on all *active* view surfaces (view surfaces must be opened before they are activated). However, input is only echoed on the view surface which is pointed to by the mouse. Most of the `Cvwsurf` fields should be zeroed, as by the `NORMAL_VWSURF` macro. Set the view surface type by assigning the *dd* (device driver) element of the *devdd* argument to the name of the appropriate device driver as in this example:[2]

```
Cvwsurf device;
NORMAL_VWSURF(device, BW2DD);
open_vws(&name, &device);
```

Note : The `NORMAL_VWSURF` macro initializes the *dd* element of the `Cvwsurf` structure and guarantees that the view surface will be opened in the normal fashion. However, to open a window with some nonstandard parameters, or open a second window from a graphics tool read the following paragraphs. To use an existing *pixwin* , then skip the following paragraphs and read Appendix F instead.

[2] Notice that when **SunCGI** specifies a pointer it usually requires that the argument is prefaced by an `&` character when the argument is actually used.

If the view surface of the specified type has been previously initialized and the type of view surface is a window (PIXWINDD or CGPIXWINDD), a CGI tool (a window with the name CGI Tool) is opened. Other characteristics of the view surface can be defined by setting the other elements of the of the *devdd* argument (which is of type `Cvwsurf`).

```
typedef struct {
    char screenname[DEVNAMESIZE]; /* physical screen */
    char windowname[DEVNAMESIZE]; /* window */
    int windowfd; /* window file descriptor */
    int retained; /* retained flag */
    int dd; /* device */
    int cmapsize; /* color map size */
    char cmapname[DEVNAMESIZE]; /* color map name */
    int flags; /* new flag */
    char **ptr; /* CGI tool descriptor */
} Cvwsurf;
```

The elements *screenname* and *windowname* specify alternate screens (for example, */dev/cgone0*) or alternate window (for example, */dev/win10*). If these elements are left blank, the current screen and the current window are used, unless the *dd* field implicitly specifies a device (for example *CGIDD*). The element *windowfd* is the window file descriptor for the current device. The current implementation of SunCGI ignores this element.

If the element *retained* is nonzero, then the view surface created by `open_vws` has a retained window associated with it (that is, if the window is covered up by another window and then revealed, the picture present before the window was covered-up will be redisplayed. By default the window created by `open_vws` is non-retained. That is, if the window is covered-up and then revealed the covered-portion will be redisplayed as white. However, drawing in non-retained windows is twice as fast as drawing in retained windows, so the choice of which type of view surface to open should be carefully considered.

The *dd* element specifies the view surface type. The *cmapsize* and the *cmapname* elements determine the size and the name of the colormap. *No colormap is enabled for monochrome devices*. The colormap determines the mapping between color indices and red, green, and blue values. If the colormap specified by the *cmapname* element of the *devdd* argument is the same as a colormap segment which already exists, then the colormap segment is shared. *cmapsize* should be a power of two, less than or equal to 256. Refer to the *SunView Programmer's Guide* for more information about colormaps.

When the *flags* element is nonzero, no attempt is made to take over the current graphics subwindow (if one exists). If this flag is set or the graphics subwindow has already been taken over by SunCGI, then a CGI Tool (a window with the name *View Surface Tool*) is created. The *ptr* element specifies the size and placement of the CGI Tool. *ptr* is a pointer to an array of characters which should consist of nine decimal numbers separated by commas. The array takes the following form:

```
"nl,nt,nw,nh,il,it,iw,ih,I"
```

Each element of the array should be filled with an integer. The first two elements specify the *x* and *y* coordinates of the upper left-hand corner of the CGI Tool. The third and fourth elements specify the width and height of the CGI Tool. The fifth through eighth elements specify the position and size of the iconic form of the CGI Tool. If the ninth element is nonzero, the tool is displayed in its iconic form.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be either in state CGOP, VSOP, or VSAC.
ENOWSTYP [11]	Specified view surface type does not exist.
EMAXVSOP [12]	Maximum number of view surfaces already open.
EMEMSPAC [110]	Space allocation has failed.
ENOTCCPW [112]	Function or argument not compatible with standard CGI.

Table 2-2 *Available View Surfaces*

Name	*Description*
PIXWINDD	SunView on a monochrome display
CGPIXWINDD	SunView on a color display
BW1DD	Full screen on a Sun-1 monochrome display
BW2DD	Full screen on a Sun-2 or Sun-3 monochrome display
CG1DD	Full screen on a Sun-1 color display
CG2DD	Full screen on a Sun-2 or Sun-3 color display
GP1DD	Full screen on a Sun-2/160 or Sun-3/160 with optional Graphics Processor

Table 2-3 *View Surface Default States*

State	*Value*
View Surface	Cleared
Device Viewport	View Surface

Note : most failures during the opening of a view surface result in error ENOWSTYP [11]. The most common reason is missetting (or failing to set) the *dd* element of the Cvwsurf structure. For example, opening a device surface type PIXWINDD instead of CGPIXWINDD on a color pixwin, or using CG2DD when the */dev/cgtwo** surface is being used by suntools. The NORMAL_VWSURF macro should be used to initialize this structure.

Activate View Surface (SunCGI Extension)

```
Cerror activate_vws(name)
Cint name; /* view surface name */
```

`activate_vws` activates the view surface specified by name. Subsequent SunCGI calls affect this view surface. Nothing is displayed on a view surface unless that view surface is active. Since a view surface is active as soon as it is opened, `activate_vws` is only need to reactivate a deactivated view surface. Note that activating a view surface may reset the state of SunCGI.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be either in state CGOP, VSOP, or VSAC.
EVSIDINV [10]	Specified view surface name is invalid.
EVSNOTOP [13]	Specified view surface not open.
EVSISACT [14]	Specified view surface is active.

Deactivate View Surface (SunCGI Extension)

```
Cerror deactivate_vws(name)
Cint name; /* view surface name */
```

`deactivate_vws` prevents calls to SunCGI functions from having an effect on this view surface. The view surface may be reactivated by `activate_vws` at a later time without having to be reopened. Note that deactivating a view surface may reset the state of SunCGI.

Errors

ENOTVSAC [4]	CGI not in proper state: CGI shall be in state VSAC.
EVSIDINV [10]	Specified view surface name is invalid.
EVSNOTOP [13]	Specified view surface not open.
EVSNTACT [15]	Specified view surface is not active.

Close View Surface (SunCGI Extension)

```
Cerror close_vws(name)
Cint name; /* view surface name */
```

`close_vws` terminates a view surface. Future SunCGI calls have no effect on this view surface. The view surface cannot be reactivated without being reopened.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be either in state CGOP, VSOP, or VSAC.
EVSIDINV [10]	Specified view surface name is invalid.
EVSNOTOP [13]	Specified view surface not open.
ENOTCCPW [112]	Function or argument not compatible with standard CGI.

Close CGI (SunCGI Extension)

```
Cerror close_cgi()
```

`close_cgi` terminates all open view surfaces, and restores the state of the SunView to the state that it was in before SunCGI was opened. Future SunCGI calls will have no effect and will generate errors.

A call to `close_cgi` should be included in the exit routines of an application program to guarantee leaving the SunView and SunCGI in a stable state.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be either in state CGOP, VSOP, or VSAC.
ENOTCCPW [112]	Function or argument not compatible with standard CGI.

2.2. View Surface Control

The functions described in this section

1. define the range of world and device coordinates,
2. control clipping, and
3. reset selected aspects of the view surface and the internal state of SunCGI.

Most functions in SunCGI express coordinates in VDC space (Virtual Device Coordinate space). In conventional computer graphics terms, VDC space corresponds to world coordinate space. The mapping between VDC space and screen space is determined by the physical size of the screen in pixels. Screen space is set by default to the entire size of the screen or the graphics window depending on the device type. The mapping from VDC space to screen space is always isotropic (the shape of the rectangle defining screen space is the same shape as VDC space). Therefore, VDC space defines the shape of the active view surface. The portion of screen space which does not correspond to VDC space is ignored. The aspect ratio (the ratio between the height and width) is therefore, defined by VDC space and not screen space.

VDC Extent

```
Cerror vdc_extent(c1, c2)
Ccoor *c1, *c2; /* bottom left-hand and */
    /* top right-hand corner of VDC space */
```

`vdc_extent` defines the limits of VDC space. The range of the coordinates must be between -32767 and 32767 (or an error is generated). VDC space can be set by the application program, but it ranges from 0 to 32767 in both the x and the y directions by default. Resetting VDC space impacts the display of output primitives on all view surfaces.

Resetting the limits of VDC space *automatically* redefines the clipping rectangle to the new limits of VDC space, regardless of the value of the *clip indicator*.

Changing the mapping from screen space to VDC space allows for translation (move) or scaling (zoom in/zoom out) of output primitives. However, no rotation functions are provided by SunCGI, and therefore, must be supplied in the application program. The code fragment below translates and zooms in on a rectangle:

```
#include <cgidefs.h>

main()
{
    Cvwsurf device;
    Cint name;
    Ccoor dv1, dv2, lower, upper;

    NORMAL_VWSURF(device, PIXWINDD);
    dv1.x = 0;
    dv1.y = 0;
    dv2.x = 200;
    dv2.y = 200;
    lower.x = 30;       /* rectangle coordinates */
    lower.y = 30;
    upper.x = 70;
    upper.y = 70;

    open_cgi();
    open_vws(&name, &device);
    vdc_extent(&dv1, &dv2);

    rectangle(&upper, &lower); /* draw initial rectangle */
    sleep(4);
    dv1.x = 0;
    dv1.y = 0;
    dv2.x = 100;
    dv2.y = 100;
    vdc_extent(&dv1, &dv2); /* center rectangle */
    rectangle(&upper, &lower);
    sleep(4);
    dv1.x = 20;
    dv1.y = 20;
    dv2.x = 80;
    dv2.y = 80;
    vdc_extent(&dv1, &dv2); /* enlarge rectangle */
    rectangle(&upper, &lower);
    sleep(20);

    close_vws(name);
    close_cgi();
}
```

Figure 2-2 *Example Program with Multiple Normalization Transformations*

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be either in state CGOP, VSOP, or VSAC.
EBADRCTD [20]	Rectangle definition is invalid.

EVDCSDIL [24]	VDC space definition is illegal.
ENOTCCPW [112]	Function or argument not compatible with standard CGI.

Device Viewport

```
Cerror device_viewport(name, c1, c2)
Cint name; /* name assigned to cgi view surface */
Ccoor *c1, *c2; /* bottom left-hand and top right-hand */
    /* corner of view surface to map device onto */
    /* (expressed in pixels) */
```

`device_viewport` redefines the limits of screen space. If the new limits are not less than or equal to the size of the current screen or window size, an error is returned. Although `device_viewport` does not redefine the aspect ratio, it may redefine which areas of the screen are unused.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be either in state CGOP, VSOP, or VSAC.
EVSIDINV [10]	Specified view surface name is invalid.
EVSNOTOP [13]	Specified view surface not open.
EBADRCTD [20]	Rectangle definition is invalid.
EBDVIEWP [21]	Viewport is not within Device Coordinates.
ENOTCCPW [112]	Function or argument not compatible with standard CGI.

Clip Indicator

```
Cerror clip_indicator(cflag)
Cclip cflag; /* CLIP, NOCLIP or CLIP_RECTANGLE */
```

For some application programs, it is desirable to clip explicitly within the viewport, while other applications may seek to increase efficiency by not checking if the coordinates are within the bounds of the clipping area.

All SunCGI application programs will run faster if clipping is turned off. However, clipping is turned on by default to prevent SunCGI from drawing outside of the bounds of the window.

The extent of VDC may be set with the `vdc_extent` function.

The value of the argument *cflag* determines whether output primitives are clipped before they are displayed. The default state is CLIP. The advantage of turning clipping off is that it improves the speed of drawing primitives. However, if clipping is set to NOCLIP, SunCGI may draw output primitives outside of the window or within the bounds of an overlapping window. If clipping is not NOCLIP, output primitives are clipped to either the clip rectangle (if *cflag* equals CLIP_RECTANGLE), or the full extent of VDC space (if *cflag* equals CLIP).

```
typedef enum {
    CLIP,
    NOCLIP,
    CLIP_RECTANGLE
} Cclip;
```

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be either in state CGOP, VSOP, or VSAC.
ENOTCCPW [112]	Function or argument not compatible with standard CGI.

Clip Rectangle

```
Cerror clip_rectangle(xmin, xmax, ymin, ymax)
Cint xmin, xmax, ymin, ymax; /* bottom left-hand */
    /* and top right-hand corner of clipping rectangle */
```

`clip_rectangle` defines the clipping rectangle in VDC Coordinates. By default, the clipping rectangle is set to the borders of VDC space. The `clip_rectangle` function defines the clipping rectangle in VDC space, to be used when clipping is set to CLIP_RECTANGLE. The clipping rectangle is automatically reset by `vdc_extent`.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be either in state CGOP, VSOP, or VSAC.
EBADRCTD [20]	Rectangle definition is invalid.
ECLIPTOL [22]	Clipping rectangle is too large.
ECLIPTOS [23]	Clipping rectangle is too small.
ENOTCCPW [112]	Function or argument not compatible with standard CGI.

Hard Reset

```
Cerror hard_reset()
```

Device control functions restore the view surface and the internal state of SunCGI to a known state. The individual aspects of the device which can be reset are the output attributes, the view surface (screen), and the error reporting.

`hard_reset` returns the output attributes to their default values; terminates all input devices, and empties the *event queue* and clears all view surfaces. VDC space is reset to its default values and the *clip indicator* is set to CLIP. This function should be used sparingly because most control, attribute, and input functions called before this function will not have any effect on functions called after `hard_reset` is called.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be either in state CGOP, VSOP, or VSAC.

Reset to Defaults

```
Cerror reset_to_defaults()
```

`reset_to_defaults` returns output attributes to defaults (see Table 4-1). `reset_to_defaults` does *not* clear the screen, reset the input devices, or reset the *character set index*.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be either in state CGOP, VSOP, or VSAC.
EVSIDINV [10]	Specified view surface name is invalid.

Clear View Surface

```
Cerror clear_view_surface(name, defflag, index)
Cint name; /* name assigned to cgi view surface */
Cflag defflag; /* default color flag */
Cint index; /* color of cleared screen */
```

`clear_view_surface` changes all pixels in the relevant area of the view surface specified by *name* to the color specified by the *index* argument, unless the *defflag* argument is set to OFF. If *defflag* is equal to OFF, the view surface is cleared to color zero. The area of the view surface which is actually cleared is determined by the `clear_control` function. `clear_view_surface` also resets the internal state of SunCGI according to previous calls to the `clear_control` function. `clear_view_surface` resets the current *background color* to the color of the cleared view surface.

Errors

ENOTVSAC [4]	CGI not in proper state: CGI shall be in state VSAC.
EVSIDINV [10]	Specified view surface name is invalid.
EVSNOTOP [13]	Specified view surface not open.
EVSNTACT [15]	Specified view surface is not active.
ECINDXLZ [35]	Color index is less than zero.
EBADCOLX [36]	Color index is invalid.

Clear Control

```
Cerror clear_control(soft, hard, intern, extent)
Cacttype soft, hard; /* soft and hard copy actions */
Cacttype intern; /* internal action */
Cexttype extent; /* clear extent */
```

`clear_control` determines the action taken when `clear_view_surface` is called. The argument *soft* can be set to either NO_OP or CLEAR. The argument *hard* which regulates clearing rules for plotters is ignored (because SunCGI does not currently support hard-copy devices) and is included only for ANSI CGI compatibility. The argument *intern* is set to either RETAIN or CLEAR. This parameter was included to support segmentation storage which is not currently a part of ANSI CGI. Therefore, the *intern* argument is ignored. The argument *extent* determines what area of the screen is cleared. It is set to one of the values in the `Cexttype` enumerated type:

```
typedef enum {
    CLIP_RECT,
    VIEWPORT,
    VIEWSURFACE
} Cexttype;
```

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be either in state CGOP, VSOP, or VSAC.
ENOTCCPW [112]	Function not compatible with CGIPW mode.

Set Error Warning Mask

```
Cerror set_error_warning_mask(action)
Cerrtype action; /* Action on receipt of an error */
```

`set_error_warning_mask`[3] determines the action taken by SunCGI when an error occurs. Three types of action are possible: NO_ACTION, POLL, INTERRUPT. If the *action* argument is set to NO_ACTION, errors are detected internally, but not reported. The error number is returned to the caller of a CGI routine. The user is advised *not* to set the *action* argument to NO_ACTION.

POLL and INTERRUPT actions print an error message on the terminal, but also return the error number (see Appendix D) so the program can perform exception handling. The default `error_warning_mask` is INTERRUPT.

Errors

ENOTOPOP [5] CGI not in proper state CGI shall be either in state CGOP, VSOP, or VSAC.

Table 2-4 *Error Warning Masks*

Error Warning Mask	*Message Printed*	*Program Aborted*	*Error Number Returned*
NO_ACTION	No	No	Yes
POLL	Yes	No	Yes
INTERRUPT	Yes	FATAL errors†	Non-FATAL errors

† SunCGI defines no errors as FATAL. All errors are non-fatal so the application has complete control to abort or perform other processing as desired. Therefore, POLL and INTERRUPT are the same in SunCGI.

2.3. Running SunCGI with SunView

SunCGI always traps five signals: SIGINT, SIGCHLD, SIGIO, SIGHUP and SIGWINCH. The first four of these cause SunCGI cleanup and program termination. When using a Graphics Processor option, SunCGI also traps SIGXCPU. Previous signal handlers, if any, are saved. When one of these signals occurs, SunCGI's signal handler will call the previous signal handler as well as performing its own processing. The actions of the previous (user installed) signal handler may interfere with SunCGI's signal responses, and are hence unsupported.

Unless a SunCGI application program has opened a *retained* view surface, overlapping another window onto a graphics subwindow will destroy the picture below. SunCGI programs can regenerate a display surface by trapping the SIGWINCH (SIGnal WINdow CHange) signal.

It is possible (though unsupported) to install a signal handler for signals after calling `open_pw_cgi` (see Appendix F). Since these signal handlers replace SunCGI's handler, the application should save SunCGI's signal handler (returned by signal), and call the saved handler when the signal occurs (amid the user's own processing). Because the response of the program to the signal then depends on the place in the user's own signal handling that SunCGI's handler is

[3] The syntax of `set_error_warning_mask` in SunCGI is slightly different from the proposed ANSI standard in that the ANSI definition allows different actions for different classes of errors.

called, results are unpredictable, and may change with a new version of SunCGI.

Note that it is not necessary for an application to catch a SIGWINCH signal, since SunCGI's `set_up_sigwinch` routine offers an easier interface. A user's `sig_function` has a different calling semantics from a SIGWINCH in that `pw_damaged` and `pw_donedamaged` have already been invoked.

When a window's contents needs regeneration during execution time, the process associated with a window receives a SIGWINCH signal. The application can use this signal to determine when a view surface needs to be regenerated. *Note*: Under no circumstances will the user be able to access the SIGWINCH signals generated when a view surface is initialized.

When a window obstructs a SunCGI view surface, output to that view surface is normally clipped to the exposed portion only (unless the clip indicator is NOCLIP). When the obstruction is removed, unless the window is RETAINED, the picture must be regenerated by re-running the output generation of the applications, for that view surface at least. An application's SIGWINCH handling function is called for this purpose.

When a SunCGI window's size changes during execution, the picture must be regenerated. But first, SunCGI updates the transformation used to map VDC space into screen space. Then, if the affected view surface is RETAINED, the retained copy is rewritten onto the view surface. (Because of the size change, this may not repair the damage satisfactorily.) Lastly, the application's SIGWINCH function is called.

Set Up SIGWINCH (SunCGI Extension)

```
Cerror set_up_sigwinch(name, sig_function)
Cint name;
Cint (*sig_function)(); /* signal handling function */
```

`set_up_sigwinch` allows the application program to trap SIGWINCH signals for view surface `name`. `sig_function` is a pointer to a function returning an integer. If `sig_function` is nonzero, all SIGWINCH signals which are not trapped by the internals of SunCGI (from view surface initialization) are passed to the function specified by `sig_function`.

The `sig_function` is called when the SIGWINCH signal is received. It is the programmer's responsibility to use a flag to determine if it is safe to process the signal at this time, or to set a flag indicating that signal processing has been put off until later. See the *SunView Programmer's Guide* for information on SIGWINCH handling.

The `sig_function` argument is called with a single argument: the name of the view surface with which it is associated by the call to `set_up_sigwinch`. This allows more than one view surface to share the same `sig_function`, and differentiate which view surface needs redisplay.

Here is an example of a program that uses `set_up_sigwinch`.

```
#include <cgidefs.h>

Ccoor box[5] = { 10000,10000 ,
                 10000,20000 ,
                 20000,20000 ,
                 20000,10000 ,
                 10000,10000 };
Cint name;
extern Cint redraw();
Cvwsurf device;

main()
{
    Ccoorlist boxlist;

    boxlist.n = 5;
    boxlist.ptlist = box;
    NORMAL_VWSURF(device, PIXWINDD);

    open_cgi();
    open_vws(&name, &device);
    set_up_sigwinch(name, redraw);

    polyline(&boxlist);
    sleep(10);

    close_vws(name);
    close_cgi();
}

Cint redraw()
{
    clear_view_surface(name, ON, 0);
}
```

Figure 2-3 *Example Program with* `set_up_sigwinch` *Function*

Errors

ENOTOPOP [5] CGI not in proper state CGI shall be either in state CGOP, VSOP, or VSAC.

2.4. Interface Negotiation

CGI is intended to support a 'negotiated device interface' which permits programs written on a specific type of hardware to run on other machines. SunCGI only allows inquiry of most of the settable modes.[4] For example the user may want to find out which types of input devices are supported. However, functions for setting color precision, coordinate type, specification mode, and color specification are *not* provided because SunCGI only supports one type of color precision (8-

[4] The functions which are not supported by SunCGI are classified as non-required by the March 1984 ANSI CGI standard. See Appendix B.

bit), coordinate type (integers), and color specification (indexed). The width and size specification modes are settable, but the functions which set them are described in Chapter 4. However, the inquiry negotiation functions are supported so that an application program written for a CGI on another manufacturers' workstation can find out whether the **SunCGI** is capable of running that application.

Inquire Device Identification

```
Cerror inquire_device_identification(name, devid)
Cint name; /* device name */
Cchar devid[DEVNAMESIZE]; /* workstation type */
```

`inquire_device_identification` reports which type of Sun Workstation view surface *name* is associated with. The argument *devid* may be set to one of the Sun Workstation types described in Table 2-2. The inclusion of the *name* argument deviates from the ANSI standard, but is necessary so that the characteristics of individual view surfaces may be inquired.

Errors

`ENOTOPOP` [5]	CGI not in proper state CGI shall be either in state CGOP, VSOP, or VSAC.
`EVSIDINV` [10]	Specified view surface name is invalid.
`EVSNOTOP` [13]	Specified view surface not open.

Inquire Device Class

```
Cerror inquire_device_class(output, input)
Cint *output, *input; /* output and input abilities */
```

`inquire_device_class` describes the capabilities of Sun Workstations in terms of the CGI functions they support.[5] Each of the two returned values reports the number of functions of each of the two classes which are supported in SunCGI. These numbers (the values of *input* and *output*) are used to make more detailed inquiries by using functions `inquire_input_capabilities` and `inquire_output_capabilities`.

Errors

`ENOTOPOP` [5]	CGI not in proper state CGI shall be either in state CGOP, VSOP, or VSAC.

Inquire Physical Coordinate System

```
Cerror inquire_physical_coordinate_system(name, xbase,
    ybase, xext, yext, xunits, yunits)
Cint name; /* name assigned to cgi view surface */
Cint *xbase, *ybase; /* base coordinates */
Cint *xext, *yext; /* pixels in x and y directions */
Cfloat *xunits, *yunits; /* number of pixels per mm. */
```

`inquire_physical_coordinate_system` reports the physical dimensions of the coordinate system of view surface *name* in pixels and millimeters. `inquire_physical_coordinate_system` is provided to permit the drawing of objects of a known physical size.

[5] The *output* argument does not include the non-standard CGI functions.

`inquire_physical_coordinate_system` is also provided to assist in the computation of parameters for the `device_viewport` function. *xext* and *yext* describe the maximum extent of the window in which the application program is run. (The window may or may not cover the entire screen.) The number of pixels per millimeter is always set to 0 because the actual screen size of device varies between individual monitors. The actual size of the screen may be obtained from the number of pixels in the *x* and *y* directions from the monitor specifications and perform the division in an application program.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be either in state CGOP, VSOP, or VSAC.
EVSIDINV [10]	Specified view surface name is invalid.
EVSNOTOP [13]	Specified view surface not open.

Inquire Output Function Set

```
Cerror inquire_output_function_set(level, support)
Cint level; /* level of output */
Csuptype *support; /* amount of support */
```

`inquire_output_function_set` reports the extent to which each level of the output portion of the ANSI CGI standard is supported.

```
typedef enum {
    NONE,
    REQUIRED_FUNCTIONS_ONLY,
    SOME_NON_REQUIRED_FUNCTIONS,
    ALL_NON_REQUIRED_FUNCTIONS
} Csuptype;
```

The standard requires that the *level* argument be an enumerated type; however, for reasons of simplicity only the level number is used by SunCGI. Levels 1-6 are supported completely (that is, both required and non-required functions are implemented. Level 7 is not supported at all. Refer to the ANSI standard for the precise definition of each level.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be either in state CGOP, VSOP, or VSAC.

Inquire VDC Type

```
Cerror inquire_vdc_type(type)
Cvdctype *type; /* type of VDC space */
```

`inquire_vdc_type` reports the type of coordinates used by SunCGI in the returned argument *type*.

```
typedef enum {
    INTEGER,
    REAL,
    BOTH
} Cvdctype;
```

type is always set to INTEGER (32-bit). SunCore is a higher-level graphics system with coordinate space expressed in real numbers.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be either in state CGOP, VSOP, or VSAC.

Inquire Output Capabilities

```
Cerror inquire_output_capabilities(first, num, list)
Cint first; /* first element */
Cint num; /* number of elements in list to be returned */
Cchar *list[]; /* returned list */
```

`inquire_output_capabilities` lists the output functions in the returned argument *list*. The range of the *first* and *num* arguments is determined by the returned argument *output* from the `inquire_device_class` function.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be either in state CGOP, VSOP, or VSAC.
EINQLTL [16]	Inquiry arguments are longer than list.

2.5. Input Capability Inquiries

Input devices have a separate class of negotiation functions. Input capability inquiries report qualitative abilities as well as quantitative abilities of input devices. The `inquire_input_capabilities` function reports which devices and overall features are supported by **SunCGI**. The remaining functions report the capabilities of individual devices or features. Input devices are virtual devices which must be *associated* with physical *triggers* (such as mouse buttons). Initializing an input device defines the measure used by a device, for example initializing a LOCATOR device defines the measure as *x-y* coordinates. In addition to being associated with a trigger, each device has selectable screen echoing capabilities. Association and echoing capabilities for each input device are reported by the functions described in this section.

Inquire Input Capabilities

```
Cerror inquire_input_capabilities(valid, table)
Clogical *valid; /* device state */
Ccgidesctab *table; /* CGI input description table */
```

`inquire_input_capabilities` reports the total number of input devices of each class that are supported. The argument *valid* returns the value L_TRUE if **SunCGI** is initialized, and L_FALSE otherwise. If *valid* is set to L_TRUE, the elements of *table* are set to the quantity and quality of inputs supported. All Sun Workstations support input at the same level.

```
typedef struct {
    Cint numloc;
    Cint numval;
    Cint numstrk;
    Cint numchoice;
    Cint numstr;
    Cint numtrig;
    Csuptype event_queue;
    Csuptype asynch;
    Csuptype coord_map;
    Csuptype echo;
    Csuptype tracking;
    Csuptype prompt;
    Csuptype acknowledgement;
    Csuptype trigger_manipulation;
} Ccgidesctab;
```

Elements of type `Cint` report how many of each type device is supported, as well as how many types of triggers are supported. Elements of type `Csuptype` report how many of the functions of each class are supported. All functions except the tracking functions are fully supported.

Errors

`ENOTOPOP` [5] CGI not in proper state CGI shall be either in state CGOP, VSOP, or VSAC.

Inquire LID Capabilities

```
Cerror inquire_lid_capabilities(devclass, devnum,
    valid, table)
Cdevoff devclass;
Cint devnum; /* device number */
Clogical *valid; /* device supported at all */
Cliddescript *table; /* table of descriptors */
```

`inquire_input_device_capabilities` describes the capabilities of a specific input device (hereafter, specified device). The input arguments *devclass* and *devnum* refer to a specific device type and number. The argument *valid* reports whether CGI is initialized.

```
typedef struct {
    Clogical    sample;
    Cchangetype change;
    Cint    numassoc;
    Cint    *trigassoc;
    Cinputability prompt;
    Cinputability acknowledgement;
    Cechotypelst *echo;
    Cchar   *classdep;
    Cstatelist  state;
} Cliddescript;
```

The elements of *table* which are of type `Clogical` indicate whether an ability is present in the specified logical input device. The *change* element reports whether associations are changeable at all (all input devices except string are changeable). The *numassoc* and *trigassoc* elements of *table* report how many

and which triggers may be associated with the specified logical input device. The *echo* argument describes which echo types are supported (see Chapter 5 for a list of echo types).[6] The *classdep* argument provides class dependent information in character form (the type of information is given in Table 2-3). If more than one piece of class dependent information is returned, then the pieces of information are separated by commas. The *state* argument reports the initial state of the specified device. See the `inquire_state_list` function.

Table 2-5 *Class Dependent Information*

Device Class	*Information*	*Possible Values*
IC_LOCATOR	Coordinate Mapping	Yes, No, Partial
	Native Range	xmin, xmax, ymin, ymax
IC_VALUATOR	Set Valuator Range	yes/no
IC_STROKE	Time Increment Settable	yes/no
	Minimum Distance	yes/no
IC_CHOICE	Range	min/max
IC_STRING	None	None

Errors

`ENOTOPOP` [5] CGI not in proper state CGI shall be either in state CGOP, VSOP, or VSAC.

Inquire Trigger Capabilities

```
Cerror inquire_trigger_capabilities(trigger, valid, tdis)
Cint trigger; /* trigger number */
Clogical *valid; /* trigger supported at all */
Ctrigdis *tdis; /* trigger description table */
```

`inquire_trigger_capabilities` describes how a particular *trigger* can be associated. The argument *valid* reports whether the device supports input at all.

```
typedef struct {
    Cchangetype change;
    Cassoclid *numassoc;
    Cint maxassoc;
    Cpromstate prompt;
    Cackstate acknowledgement;
    Cchar *name;
    Cchar *description;
} Ctrigdis;
```

The *change* element of *tdis* reports whether the specified trigger can be associated with a logical input device. The *numassoc* element of *tdis* gives supported LID associations for this trigger. This consists of *n*, the number of LID classes which can be associated with the trigger, a pointer to an array of *n* entries telling which *n* device classes can be associated with the trigger, and how many of each

[6] Note that `inquire_lid_capabilities` returns an enumerated type whereas `track_on` accepts integers. Therefore these values may be different.

device class is defined. The *maxassoc* field gives the number of LID's which can be concurrently associated with this trigger. SunCGI does not support either prompt or acknowledgement for any input device. The *name* element is simply a character form of the trigger name (for example, LEFT MOUSE BUTTON). The *description* element is never filled and is included for standards compatibility.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be either in state CGOP, VSOP, or VSAC.
EINTRNEX [86]	Trigger does not exist.

3

Output

3

Output

SunCGI supports two classes of output primitives: geometrical output primitives and raster primitives.

Geometrical Output Primitives
include arcs, circles, polylines, and polygons. The position of geometrical output primitives are always specified in absolute VDC coordinates.[7]

Raster Primitives
draw text and scaled and unscaled 2D arrays. The coordinate system for raster primitives depends on the type of primitive. The drawing mode determines how output primitives are drawn on top of other output primitives or the background.

3.1. Geometrical Output Primitives

Geometrical output primitives are divided into two classes: polygonal primitives and conical primitives. Geometrical output primitives are all 2D in keeping with the CGI standard. However, polygons with holes (via the `partial_polygon` function) are provided in order to support 3D graphics packages.

Geometrical primitives (except `polymarker`) are considered either closed or not closed. Polymarker uses its own attributes (see Section 4.3). Non-closed figures (polylines, circular arcs, or elliptical arcs) are drawn with a style, width and color determined from line attributes (see Section 4.2). Closed figures (polygons, rectangles, circles, ellipses, and circular and elliptical closed arcs) use the solid object attributes (see Section 4.4). The geometrical information specifies the boundary of a closed figure. The interior of this boundary is filled using fill area attributes. The boundary may be surrounded with a line, drawn with perimeter attributes, not the line attributes. For example, a circle of radius 1000 and a perimeter width of 100 VDC units has its perimeter between the circle of radius 1000 and a concentric circle of radius 1100 (not from 950 through 1050).

Most polygonal primitives `polyline`, (`polymarker`, `polygon`, and `partial_polygon`) take one argument of type `Ccoorlist`:

[7] SunCGI (unlike SunCore) maintains no concept of current position.

```
typedef struct {
    Cint x;
    Cint y;
} Ccoor;

typedef struct {
    Ccoor *ptlist;
    Cint n;
} Ccoorlist;
```

The element *ptlist* is really a pointer to an array of type `Ccoor` which contains the *n* coordinates of the points defining the primitive. The style, color, and other features of lines, markers, and fill patterns used by geometrical output primitives are set by the attribute functions described in Chapter 4.

The polygons generated by **SunCGI** may or may not be closed. **SunCGI** automatically assumes the polygon is closed for the purpose of filling. However, a polygon must be explicitly closed in order to get all of its edges drawn, so take care to generate explicitly closed polygons. The `rectangle` function implicitly generates closed objects.[8]

SunCGI has two classes of conical primitives: *circular* and *elliptical*. Each class has functions for drawing solid objects, arcs, and closed arcs. Drawing of conical primitives is regulated by the same attributes that regulate the drawing of polygons and polylines.

Polyline

```
Cerror polyline(polycoors)
Ccoorlist *polycoors; /* list of points */
```

`polyline` draws lines between the points specified by the *ptlist* element of *polycoors*. `polyline` does *not* draw a line between the first and last element of the point list. To generate a closed polyline, the last point on the list must have the same coordinates as the first point on the list. The style, color, and width of the lines are set by the `polyline_bundle_index`, `line_type`, `line_color`, `line_width` and `line_width_specification_mode` functions. If a line segment of a polyline has a length of zero, the line is not drawn. To draw a point, use the `circle` function. If you specify a polyline that has less than two points, an error is generated. Similarly, if the number of points specified is greater than the maximum number of points (MAXPTS) an error is generated.

Errors

`ENOTVSAC` [4]	CGI not in proper state: CGI shall be in state VSAC.
`ENMPTSTL` [60]	Number of points is too large.
`EPLMTWPT` [61]	polylines must have at least two points.

Disjoint Polyline

[8] A closed portion of a closed figure boundary will not be drawn if it exceeds a clipping boundary.

```
Cerror disjoint_polyline(polycoors)
Ccoorlist *polycoors; /* list of points */
```

`disjoint_polyline` draws lines between pairs of elements in *ptlist*. The line attributes described in Section 4.2 determine the appearance of the `disjoint_polyline` function. If *polycoors* contains an odd number of points, the last point is ignored. As with `polyline`, if the number of points is less than two or greater than MAXPTS, an error is generated. `disjoint_polyline` is typically used to implement scan-line polygon filling algorithms.

Errors

`ENOTVSAC` [4]	CGI not in proper state: CGI shall be in state VSAC.
`ENMPTSTL` [60]	Number of points is too large.
`EPLMTWPT` [61]	polylines must have at least two points.

Polymarker

```
Cerror polymarker(polycoors)
Ccoorlist *polycoors; /* list of points */
```

`polymarker` draws a marker at each point. The type, color, and size of marker are set by the `polymarker_bundle_index`, `marker_type`, `marker_color`, `marker_size`, and `marker_size_specification_mode` functions. If the number of points specified is greater than the maximum number of points, an error is generated. `polymarker` is useful for making graphs such as scatter plots.

Errors

`ENOTVSAC` [4]	CGI not in proper state: CGI shall be in state VSAC.
`ENMPTSTL` [60]	Number of points is too large.

Polygon

```
Cerror polygon(polycoors)
Ccoorlist *polycoors; /* list of points */
```

`polygon` displays the polygon described by the points in *polycoors*. In addition, any points added to the *global polygon list* by the `partial_polygon` function are also displayed. The polygon is filled between edges. Polygons are allowed to be self-intersecting. The visibility of individual edges can only be set by the `partial_polygon` function. The style and color used to fill the polygon are set by the solid object attribute functions described in Chapter 4. The characteristics of the edges are controlled by the perimeter attribute functions. The number of points in the polygon used to determine the error condition of too few or too many points is the total number of points on the *global polygon list*, not the number of points specified in *polycoors*. After the polygon is drawn, the *global polygon list* is emptied.

Errors

`ENOTVSAC` [4]	CGI not in proper state: CGI shall be in state VSAC.
`ENMPTSTL` [60]	Number of points is too large.
`EPGMTHPT` [62]	Polygons must have at least three points.

EGPLISFL [63] Global polygon list is full.

Partial Polygon

```
Cerror partial_polygon(polycoors, cflag)
Ccoorlist *polycoors; /* list of points */
Ccflag cflag; /* CLOSE previous polygon? */
```

`partial_polygon` adds elements to the *global polygon list* without displaying the polygon. The `partial_polygon` function provides the capability of drawing multiple-boundary polygons, including polygons with holes. The drawing is actually performed when `polygon` is called. `polygon` will close the last boundary on the *global polygon list* and add the coordinate list it is passed as the final polygon boundary before drawing.

cflag controls whether the last polygon in the *global polygon list* is open or closed. If *cflag* is set to CLOSE, the last polygon on the *global polygon list* will be closed by drawing a *visible* perimeter edge between the last and the first points of the last polygon on the *global polygon list*. If the *cflag* is set to OPEN, the points in *polycoors* are appended to the last polygon on the *global polygon list*, but an *invisible* perimeter edge will be drawn between the last point currently on the *global polygon list* and the first point in the `Ccoorlist`. The visibility of polygon edges can be individually controlled by calling `partial_polygon` with *cflag* set to OPEN for each invisible edge and with *cflag* set to CLOSE for each new boundary. The interpretation of *cflag* is slightly different than the pseudocode given in the CGI standard. Future versions of CGI may use a different syntax to offer the capabilities of multiple-boundary polygons and invisible edges.

The CGI standard specifies that `circle`, `rectangle`, `ellipse` and `close_arc` are primitives that may use the *global polygon list* for filling. SunCGI does not use the *global polygon list* in these functions, and therefore leaves it untouched. These SunCGI routines *do not* empty the *global polygon list*.

```
#include <cgidefs.h>

main()
{
    Ccoor list[4];
    Ccoorlist points;
    Cint name;
    Cvwsurf device;

    NORMAL_VWSURF(device, PIXWINDD);

    open_cgi();
    open_vws(&name, &device);

    interior_style(SOLIDI, ON);
    list[0].x = 10000;
    list[0].y = 10000;
    list[1].x = 10000;
    list[1].y = 20000;
    list[2].x = 20000;
    list[2].y = 20000;
    list[3].x = 20000;
    list[3].y = 10000;
    points.ptlist=list;
    points.n=4;
    partial_polygon(&points, CLOSE);
    list[0].x = 12500;
    list[0].y = 12500;
    list[1].x = 12500;
    list[1].y = 17500;
    list[2].x = 17500;
    list[2].y = 17500;
    list[3].x = 17500;
    list[3].y = 12500;
    points.ptlist=list;
    points.n=4;
    polygon(&points); /* cut a hole in it */

    sleep(10);

    close_vws(name);
    close_cgi();
}
```

Figure 3-1 *Example Program with Polygons*

An error is detected if the number of points on the *global polygon list* exceeds MAXPTS. In this case, the polygon on the *global polygon list* is drawn, and the new information is not added. The same error handling applies to `polygon`.

Errors

ENOTVSAC [4]	CGI not in proper state: CGI shall be in state VSAC.
ENMPTSTL [60]	Number of points is too large.
EPGMTHPT [62]	Polygons must have at least three points.
EGPLISFL [63]	Global polygon list is full.

Rectangle

```
Cerror rectangle(rbc, ltc)
Ccoor *rbc, *ltc; /* corners defining rectangle */
```

`rectangle` displays a box with its lower right-hand corner at point *rbc* and its upper left-hand corner at point *ltc*. Calls to `rectangle` do not affect the *global polygon list*. The interior of the rectangle (the filled portion) is defined by *rbc* and *ltc*. The perimeter is drawn outside of this region. The appearance of the rectangle is determined by the fill area and perimeter attributes. A rectangle with one side coincident with a clipping boundary specifies an interior extending to the boundary. Hence, a portion of the perimeter is outside the clipping boundary and is not drawn.

If the arguments to `rectangle` would result in a point or a line, the point or line is drawn. However, if the arguments to rectangle determine a point, the point is drawn with width zero, regardless of the current value of *perimeter width*. If the values of *rbc* and *ltc* are reversed, the points are automatically reversed and the rectangle is drawn normally.

Errors

ENOTVSAC [4]	CGI not in proper state: CGI shall be in state VSAC.

Circle

```
Cerror circle(cl, rad)
Ccoor *cl; /* center */
Cint rad; /* radius */
```

`circle` draws a circle of radius *rad* centered at *cl*. The argument *rad* is expressed in terms of VDC space. The color, form, and visibility of the interior and perimeter are controlled by the same solid object attributes which control the drawing of polygons and rectangles.

The argument *rad* determines the size of the *interior* of the circle. Therefore, a circle with a thick perimeter may be larger than expected. If the radius is zero, a point is drawn, and no textured perimeter is drawn, even if the perimeter width is large. If the radius is negative, the absolute value of the radius is used.

Textured circles may possibly contain an incorrect element at one point because the digital circumference may not be exactly divisible by the length of the texture element.

Errors

ENOTVSAC [4]	CGI not in proper state: CGI shall be in state VSAC.

Circular Arc Center

```
Cerror circular_arc_center(cl, c2x, c2y, c3x, c3y, rad)
Ccoor *cl; /* center */
Cint c2x, c2y, c3x, c3y; /* endpoints */
Cint rad; /* radius */
```

`circular_arc_center` draws a circular arc between points *c2x*, *c2y* and *c3x*, *c3y* with circle of radius *rad* at center *c1*. Point *c2x*, *c2y* is the starting point and point *c3x*, *c3y* is the ending point. Circular arcs are drawn in a *counterclockwise* manner. This convention is used to determine the difference between the arc formed by the smaller angle determined by *c2x*, *c2y*, *c1* and *c3x*, *c3y* and the larger angle specified by these same points. Therefore switching the values of *c2x*, *c2y* and *c3x*, *c3y* will produce arcs which total 360 degrees. If *rad* is negative, the points 180 degrees opposite from *c2x*, *c2y* and *c3x*, *c3y* are used as the endpoints of the arc.

If the *rad* is zero, a point is drawn at *c1*. If either *c2x*, *c2y* or *c3x*, *c3y* are not on the circumference of the circle determined by *c1* and *rad*, an error is generated and the arc is not drawn. The attributes which determine the style, width, and color of the arc are the same functions which regulate the drawing of *polylines*.

Errors

`ENOTVSAC` [4]	CGI not in proper state: CGI shall be in state VSAC.
`EARCPNCI` [64]	Arc points do not lie on circle.

Circular Arc Center Close

```
Cerror circular_arc_center_close(c1, c2x,
    c2y, c3x, c3y, rad, close)
Ccoor *c1; /* center */
Cint c2x, c2y, c3x, c3y; /* endpoints */
Cint rad; /* radius */
Cclosetype close; /* PIE or CHORD */
```

`circular_arc_center_close` draws a closed arc centered at *c1* with radius *rad* and endpoints *c2x*, *c2y* and *3x*, *c3y*. Arcs are closed with either the PIE or CHORD algorithm. The PIE algorithm draws a line from each of the endpoints of the arc to the center point of the circle. **SunCGI** then fills this region as it would any other solid object. The CHORD algorithm draws a line connecting the endpoints of the arc and then fills this region using solid object attributes. `circular_arc_center_close` is useful for drawing pie charts (see following example):

```
#include <cgidefs.h>

main() /* draws four quadrants in different colors */
{
    Ccoor c1;
    Cint name, radius;
    Cvwsurf device;

    c1.x = 16000;   /* center */
    c1.y = 16000;
    NORMAL_VWSURF(device, CGPIXWINDD);
    radius = 8000;  /* radius */

    open_cgi();
    open_vws(&name, &device);

    interior_style(SOLIDI, OFF);
    fill_color(1);      /* color of quadrant 1 */
    circular_arc_center_close(&c1, 24000, 16000,
        16000, 24000, radius, PIE);
    fill_color(2);      /* color of quadrant 2 */
    circular_arc_center_close(&c1, 16000, 24000,
        8000, 16000, radius, PIE);
    fill_color(3);      /* color of quadrant 3 */
    circular_arc_center_close(&c1, 8000, 16000,
        16000, 8000, radius, PIE);
    fill_color(4);      /* color of quadrant 4 */
    circular_arc_center_close(&c1, 16000, 8000,
        24000, 16000, radius, PIE);

    sleep(10);
    close_vws(name);
    close_cgi();
}
```

Figure 3-2 *Example Program with Four Circle Quadrants in Different Colors*

Errors

ENOTVSAC [4]	CGI not in proper state: CGI shall be in state VSAC.
EARCPNCI [64]	Arc points do not lie on circle.

Circular Arc 3pt

```
Cerror circular_arc_3pt(c1, c2, c3)
Ccoor *c1, *c2, *c3; /* starting,
    intermediate and ending points */
```

`circular_arc_3pt` draws a circular arc starting at point *c1* and ending at point *c3* which is *guaranteed* to pass through point *c2*. The line attributes functions described in Section 4.2 determine the appearance of the `circular_arc_3pt` function. If the circular arc is textured (for example, dotted) then the intermediate point may not be displayed. However, if the arc is solid, the intermediate point is always drawn. If the three points are colinear, a

line is drawn. If two of the three points are coincident, a line is drawn between the two distinct points. Finally, if all three points are coincident, a point is drawn. `circular_arc_3pt` is considerably slower than `circular_arc_center`, therefore, you are advised to `circular_arc_center` if both functions can meet your needs.

Errors

ENOTVSAC [4] CGI not in proper state: CGI shall be in state VSAC.

Circular Arc 3pt Close

```
Cerror circular_arc_3pt_close(c1, c2, c3, close)
Ccoor *c1, *c2, *c3; /* starting, intermediate
    and ending points */
Cclosetype close; /* PIE or CHORD */
```

`circular_arc_3pt_close` draws a circular arc starting at point *c1* and ending at point *c3* which is guaranteed to pass through point *c2*. The solid object attributes described in Section 4.4 determine the appearance of the `circular_arc_3pt_close` function. As with `circular_arc_3pt`, `circular_arc_3pt_close` is considerably slower than `circular_arc_center_close`; therefore, you are advised to use `circular_arc_center_close` if both functions meet your needs.

If the three points are colinear, a line is drawn. If two of the three points are coincident, a line is drawn between the two distinct points. Finally, if all three points are coincident, a point is drawn. In none of these cases will any region be filled.

Errors

ENOTVSAC [4] CGI not in proper state: CGI shall be in state VSAC.

Ellipse

```
Cerror ellipse(c1, majx, miny)
Ccoor *c1; /* center */
Cint majx, miny; /* length of x and y axes */
```

ellipse draws an ellipse centered at point *c1* with major (x) and minor (y) axes of length *majx* and *miny*.[9] If either *majx* or *miny* are zero, a line is drawn. If both *majx* and *miny* are zero, a point is drawn. The attributes which control the drawing of ellipses are the solid object attributes described in Section 4.4.

Errors

ENOTVSAC [4] CGI not in proper state: CGI shall be in state VSAC.

Elliptical Arc

```
Cerror elliptical_arc(c1, sx, sy, ex, ey, majx, miny)
Ccoor *c1;/* center */
Cint sx, sy; /* starting point of arc */
Cint ex, ey; /* ending point of arc */
Cint majx, miny; /* endpoints of major and minor axes */
```

`elliptical_arc` draws an elliptical arc centered at *c1* with major (x) and minor (y) axes of length *majx* and *miny*. *sx*, *sy* and *ex*, *ey* are the starting and

[9] Although the axes are called the major and minor axes by the standard they are really the x and y axes. In fact, the x axis can either be the major or minor axis, depending on the relative length of the y axis.

ending points of the arc. An error is generated (and the ellipse is not drawn) if the points (*sx*, *sy*, and *ex*, *ey*) are not on the perimeter of the ellipse. Elliptical arcs are drawn in a *counterclockwise* manner. This convention is used to determine the difference between the arc formed by the obtuse angle determined by *c1.x*, *c1.y*, *sx*, *sy*, and *ex*, *ey* and the acute angle specified by these same points. Therefore switching the values of *sx*, *sy* and *ex*, *ey* will produce complementary arcs.

If either *majx* or *miny* are zero, a line is drawn. If both *majx* and *miny* are zero, a point is drawn. Polyline attributes are used to determine the appearance of elliptical arcs.

Errors

ENOTVSAC [4] CGI not in proper state: CGI shall be in state VSAC.

EARCPNEL [65] Arc points do not lie on ellipse.

Elliptical Arc Close

```
Cerror elliptical_arc_close(c1, sx, sy, ex,
    ey, majx, miny, close)
Ccoor *c1;/* center */
Cint sx, sy; /* starting point of arc */
Cint ex, ey; /* ending point of arc */
Cint majx, miny; /* endpoints of major and minor axes */
Cclosetype close; /* PIE or CHORD */
```

`elliptical_arc_close` draws an elliptical arc specified by *sx*, *sy*, *ex*, *ey* and *majx*, *miny*. The arc is closed with either the PIE or CHORD algorithm. The same restrictions on *sx*, *sy*, *ex*, and *ey* are applied to `elliptical_arc_close` as to `elliptical_arc`. However, `elliptical_arc_close` uses the fill area and perimeter attributes, whereas `elliptical_arc` uses the line attributes.

If either *majx* or *miny* are zero, a line is drawn. If both *majx* and *miny* are zero, a point is drawn. In neither of these cases will any region be filled.

Errors

ENOTVSAC [4] CGI not in proper state: CGI shall be in state VSAC.

EARCPNEL [65] Arc points do not lie on ellipse.

3.2. Raster Primitives

Raster primitives include text, cell arrays, pixel arrays, and bitblts (bit block transfer). Bitblts are pixel arrays (bitmaps) which can be drawn using the various drawing modes. The current *drawing mode* determines how bitblt primitives are affected by information which is already on the screen. Raster primitives differ from geometrical primitives because their dimensions are not necessarily expressed in VDC space. Therefore, you must be careful to consider whether position arguments are expressed in VDC space or screen coordinates.

Text

```
Cerror text(c1, tstring)
Ccoor *c1; /* starting point of text (in VDC space) */
Cchar *tstring; /* text */
```

`text` displays the text contained in *tstring* at point *c1* (expressed in VDC space). The appearance of text is controlled by the text attributes described in Section

4.8. Control characters are displayed as blanks, except in the SYMBOL font where they may be drawn as pictures of bugs.

Errors

ENOTVSAC [4] CGI not in proper state: CGI shall be in state VSAC.

VDM Text

```
Cerror vdm_text(cl, flag, tstring)
Ccoor *cl; /* starting point of text (in VDC space) */
Ctextfinal flag; /* final text for alignment */
Cchar *tstring; /* text */
```

`vdm_text` displays the text contained in *tstring* at point *cl* (expressed in VDC space). The intended difference between `text` and `vdm_text` is that `vdm_text` allows control characters; however, SunCGI does not handle control characters so text drawn with `vdm_text` will appear identical to text drawn with the `text` function. If the *flag* argument is equal to FINAL, the previous text and the appended text are aligned separately. However, if the *flag* argument is equal to NOT_FINAL, the appended and previous text are aligned together.

Errors

ENOTVSAC [4] CGI not in proper state: CGI shall be in state VSAC.

Append Text

```
Cerror append_text(flag, tstring)
Ctextfinal flag; /* final text for alignment */
Cchar *tstring; /* text */
```

`append_text` displays the text contained in *tstring* after the end of the most recently written text. The type of text written depends on the same attributes which control the display of text. The *flag* argument determines whether the appended text is aligned with the previous text if the alignment is CONTINUOUS. If the *flag* argument is equal to FINAL, then the previous text and the appended text are aligned separately. However, if the *flag* argument is equal to NOT_FINAL, the appended and previous text are aligned together.

Errors

ENOTVSAC [4] CGI not in proper state: CGI shall be in state VSAC.

Inquire Text Extent

```
Cerror inquire_text_extent(tstring, nextchar, concat,
    lleft, uleft, uright)
Cchar *tstring; /* text */
Cchar nextchar; /* next character (for kerning) */
Ccoor *concat; /* concatenation point */
Ccoor *lleft, *uleft, *uright;
    /* coordinates of text bounding box */
```

`inquire_text_extent` determines how large text *tstring* would be and where it would be placed if it were drawn using the current text attributes. The *nextchar* parameter is used to determine the point where text would start if more text (starting with *nextchar*) were appended to the text specified by *tstring*.[10] If *nextchar* equals '*single space*', the last point of the current character is used. The argument *concat* returns the coordinates of the point where appended text

[10] This is a method for accounting for proportional spacing.

would start. The arguments *lleft*, *uleft*, and *uright* return three of the four corners of the bounding box of text contained in *tstring*.

The bounding box is a parallelogram (a rectangle if the character up vector and the character base vector are orthogonal). The names of the parallelogram corners are correct if no rotation is applied to the text. For some character orientations, the implied relationships do not hold. For example, *lleft* may not be the lowest. The fourth corner may be easily calculated from the three returned:

```
uright->x + lleft->x - uleft->x
uright->y + lleft->y - uleft->y
```

The concatenation point and text alignment parallelogram are returned in VDC space, but assume a text position of (0, 0). If the text is to be drawn at a position (x, y) then (x, y) must be added to each point to yield the true locations.

The values of *lleft*, *uleft*, and *uright* are defined by the bounding box of the character and therefore may not be at the exact pixel where the character ends or begins.

Errors

ENOTVSAC [4] CGI not in proper state: CGI shall be in state VSAC.

Cell Array

```
Cerror cell_array(p, q, r, dx, dy, colorind)
Ccoor *p, *q, *r;
    /* corners of parallelogram (in VDC space) */
Cint dx, dy; /* dimensions of color array */
Cint *colorind; /* array of color values */
```

`cell_array` draws a scaled and skewed pixel array on the view surface(s). Points p, q and r (expressed in VDC space) define a parallelogram. Line p-q is a diagonal and p is the lower left-hand corner. r is one of the remaining two corners. *dx* and *dy* define the width and the height of the array *colorind* which is mapped onto the parallelogram defined by p, q, and r.

`cell_array` is one of the few primitives which depends on the actual size of the view surface. Cell arrays are not drawn if the elements of the array would be smaller than one pixel. However, because different view surfaces may have different dimensions, a cell array might be drawn on one view surface, but not on another smaller view surface. Finally, all cells composing the cell array are the same size; therefore, the upper left hand corner of the cell array might be down and to the right of point q because of the accumulated error of making all of the cells slightly smaller than their floating point size. For example if each cell of a 3×3 cell array is supposed to be 3.333 pixels wide, the actual cell array will be nine pixels wide instead of ten.

Errors

ENOTVSAC [4] CGI not in proper state: CGI shall be in state VSAC.

ECELLATS [66] Cell array dimensions *dx*, *dy* are too small.

ECELLPOS [67] Cell array dimensions must be positive.

Pixel Array

```
Cerror pixel_array(pcell, m, n, colorind)
Ccoor *pcell; /* base of array in VDC space */
Cint m, n; /* dimensions of color array in screen space */
Cint *colorind; /* array of color values */
```

`pixel_array` draws array *colorind* starting at point *pcell* (expressed in VDC space). *m* and *n* (expressed in screen space) define the *x* and *y* dimensions of the array. Therefore, *pixel arrays* always have a constant physical size, independent of the dimensions of VDC space. The *pixel array* is drawn *down* and to the *right* from point *pcell*. If either *m* or *n* are not positive, the absolute value of *m* and *n* are used. `pixel_array` is *not* affected by the current *drawing mode*.

Errors

`ENOTVSAC` [4] CGI not in proper state: CGI shall be in state VSAC.

`EVALOVWS` [69] Value outside of view surface.

BitBlt Source Array

```
Cerror bitblt_source_array(pixsource, xo, yo, xe, ye,
    pixtarget, xt, yt, name)
Cpixrect *pixsource, *pixtarget;
    /* source and target pixel arrays */
Cint xo, yo;
    /* coordinates of source array (in VDC space) */
Cint xe, ye;
    /* dimensions of source array (in screen space) */
Cint xt, yt;
    /* coordinates of target pixel array (in VDC space) */
Cint name; /* view surface name */
```

`bitblt_source_array` moves a pixel array from point (*xo*, *yo*) to point (*xt*, *yt*) using the current *drawing mode*. Both of these points are expressed in VDC space. The size of the pixel array is determined by the *xe* and *ye* arguments which are expressed in screen space. *pixsource* and *pixtarget* are pointers to *pixrects* which must already be created by `mem_create`.[11] These pixrects must be the same depth as the view surface: 1-bit deep on a monochrome device, 8-bit on a color device. The source area of the view surface associated with name is saved into `pixsource` (at 0,0). The target area, after `pixsource` is applied to it, is read into `pixtarget` pixrect (at 0,0).

An error is detected if either *xe* or *ye* are not positive. If the replicated pattern array overlaps with the source array on the screen, the visual result depends on the current *drawing mode*. *pixsource* and *pixtarget* may have different contents depending on the screen drawing mode (see the `set_drawing_mode` function).

Multiple view surfaces and bitblt's are incompatible, so a *name* argument must be specified.

Errors

`ENOTVSAC` [4] CGI not in proper state: CGI shall be in state VSAC.

[11] Refer to the *Pixrect Reference Manual* for more information about pixrects.

EVALOVWS [69] Value outside of view surface.

BitBlt Pattern Array

```
Cerror bitblt_pattern_array(pixpat, px, py, pixtarget,
    rx, ry, ox, oy, dx, dy, name)
Cpixrect *pixpat; /* pattern source array */
Cint px, py; /* pattern extent */
Cpixrect *pixtarget; /* destination pattern array */
Cint rx, ry; /* pattern reference point */
Cint ox, oy; /* destination origin */
Cint dx, dy; /* destination extent */
Cint name; /* view surface name */
```

`bitblt_pattern_array` replicates the pattern (using the current *drawing mode*) stored in *pixpat* to fill the area of the view surface which is determined by *ox*, *oy* and *dx*, *dy*. The pattern reference point determines the offset of the pattern array from the point zero. The resultant pattern array is displayed at *ox*, *oy*. The visual result depends on the current drawing mode.

`pixpat` is a pointer to a pixrect which must be created and initialized with the pattern by the application program. `pixtarget` is a pointer to a pixrect (with same depth as the device) which must already be created by the user, using `mem_create`. The target area, after `pixpat` is applied to it, is read into the `pixtarget` pixrect (at 0,0).

Multiple view surfaces and bitblt's are incompatible, so a *name* argument must be specified.

Errors

ENOTVSAC [4] CGI not in proper state: CGI shall be in state VSAC.

EVALOVWS [69] Value outside of view surface.

EPXNOTCR [70] Pixrect not created.

BitBlt Patterned Source Array

```
Cerror bitblt_patterned_source_array(pixpat, px, py,
    pixtarget, rx, ry, pixsource, sx, sy, ox, oy,
    dx, dy, name)
Cpixrect *pixpat; /* pattern source array */
Cint px, py; /* pattern extent */
Cpixrect *pixsource; /* source array */
Cint sx, sy; /* source origin */
Cpixrect *pixtarget; /* destination pattern array */
Cint rx, ry; /* pattern reference point */
Cint ox, oy; /* destination origin */
Cint dx, dy; /* destination extent */
Cint name; /* view surface name */
```

`bitblt_patterned_source_array` replicates (using the current drawing mode) the pattern stored in `pixpat` to fill the area of the view surface determined by ox, oy and dx, dy. The source area of the view surface is read into the pixrect pointed to by `pixsource` (which must already be created by the user with same depth as the device) at 0,0. The source area is stenciled through the replicated pattern onto the view surface at ox, oy, using the current drawing mode. The target area, after the copy, is read into the `pixtarget` pixrect. If

the replicated pattern array overlaps with the source array on the screen, the visual result depends on the current drawing mode.

Multiple view surfaces and bitblt's are incompatible, so a *name* argument must be specified.

Errors

ENOTVSAC [4]	CGI not in proper state: CGI shall be in state VSAC.
EVALOVWS [69]	Value outside of view surface.
EPXNOTCR [70]	Pixrect not created.

Inquire Cell Array

```
Cerror inquire_cell_array(name, p, q, r, dx, dy, colorind)
Cint name; /* view surface name */
Ccoor *p, *q, *r;
    /* corners of parallelogram (in VDC space) */
Cint dx, dy; /* dimensions of color array */
Cint *colorind; /* array of color values */
```

Points *p*, *q* and *r* (in VDC space) define a parallelogram with line *p-q* as the diagonal where *p* is the lower left-hand corner. *r* is one of the remaining two corners. *dx* and *dy* define the width and the height of the array *colorind* which contains the colors of the pixels on the screen which lie within the parallelogram defined by *p*, *q*, and *r*. Notice that a view surface identifier, *name*, must be specified because the result of this function is highly dependent on the dimensions and contents of the view surface.

The area of the screen corresponding to the parallelogram is assumed to contain a regular grid of points. However, if each element of the grid is larger than one pixel, the color of the pixel at lower left-hand corner of each element of the grid is defined to be the color of the grid element. Therefore, the values contained in *colorind* are highly dependent on the size of the view surface. An error is produced if the elements of the grid are smaller than one pixel.

Errors

ENOTVSAC [4]	CGI not in proper state: CGI shall be in state VSAC.
EVSIDINV [10]	Specified view surface name is invalid.
EVSNOTOP [13]	Specified view surface not open.
EVSNTACT [15]	Specified view surface is not active.
ECELLATS [66]	Cell array dimensions dx, dy are too small.
ECELLPOS [67]	Cell array dimensions must be positive.

Inquire Pixel Array

```
Cerror inquire_pixel_array(p, m, n, colorind, name)
Ccoor *p; /* base of array in VDC space */
Cint m, n; /* dimensions of color array in screen space */
Cint *colorind; /* array of color values */
Cint name; /* view surface name */
```

`inquire_pixel_array` fills array *colorind* with the values of pixels in the area of the screen defined by point *p* (expressed in VDC space) and *m* and *n* (expressed in screen space). The array is filled *down* and to the *right* from point

p. If either *m* or *n* are not positive, the absolute value of these arguments is used.

Multiple view surfaces and bitblt's are incompatible, so a *name* argument must be specified.

Errors

ENOTVSAC [4]	CGI not in proper state: CGI shall be in state VSAC.
EVALOVWS [69]	Value outside of view surface.
EPXNOTCR [70]	Pixrect not created.

Inquire Device Bitmap

```
Cpixrect *inquire_device_bitmap(name)
Cint name; /* name assigned to cgi view surface */
```

`inquire_device_bitmap` returns the pixrect which corresponds to the view surface. The pixrect describes the entire device, even if the view surface is a smaller pixwin. If you want to use subareas of this pixrect or manipulate it any other way, refer to the *Pixrect Reference Manual*.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in in state VDOP, VSOP, or VSAC.

Inquire BitBlt Alignments

```
Cerror inquire_bitblt_alignments(base, width, px, py,
    maxpx, maxpy, name)
Cint *base; /* bitmap base alignment */
Cint *width; /* width alignment */
Cint *px, *py; /* pattern extent alignment */
Cint *maxpx, *maxpy; /* maximum pattern size */
Cint name; /* name assigned to cgi view surface */
```

`inquire_bitblt_alignments` reports the alignment criteria which are necessary for some implementations. These factors are not critical for SunCGI. However, you should keep in mind the appropriate depth for the pixrect when talking to a specific device. Therefore the arguments *base*, *width*, *px*, and *py* are always set to zero. The arguments *maxpx* and *maxpy* are device dependent and determine the maximum size of a pattern for `bitblt_pattern_array` and `bitblt_patterned_source_array`.

Multiple view surfaces and bitblt's are incompatible, so a *name* argument must be specified.

Errors

ENOTVSAC [4]	CGI not in proper state: CGI shall be in state VSAC.
EVSIDINV [10]	Specified view surface name is invalid.
EVSNOTOP [13]	Specified view surface not open.
EVSNTACT [15]	Specified view surface is not active.

3.3. Drawing Modes

Drawing modes determine the result of drawing any output primitive on the clear screen (background) or on top of a previously drawn object. Drawing modes only affect the drawing of *bitblt* primitives. However, a non-standard

`set_global_drawing_mode` function is provided, which affects all output primitives *except* bitblt's. Resetting the drawing mode in the middle of an application program only affects those output primitives drawn after the mode is reset. The novice user is advised *not* to reset the drawing mode until the user has written at least one application program using SunCGI.

Set Drawing Mode

```
Cerror set_drawing_mode(visibility, source,
    destination, combination)
Cbmode visibility; /* transparent or opaque */
Cbitmaptype source; /* NOT source bits */
Cbitmaptype destination; /* NOT destination bits */
Ccombtype combination; /* combination rules */
```

`set_drawing_mode` determines the current *drawing mode* which in turn determines how bitblt primitives are displayed. The *visibility* argument determines how pixels with index zero are treated.

```
typedef enum {
    TRANSPARENT,
    OPAQUE
} Cbmode;

typedef enum {
    BITTRUE,
    BITNOT
} Cbitmaptype;

typedef enum {
    REPLACE,
    AND,
    OR,
    NOT,
    XOR
} Ccombtype;
```

If *visibility* is set to TRANSPARENT, all source pixels with index zero leave the destination pixel unchanged, regardless of the operation, whereas if *visibility* is set to OPAQUE, all pixels are treated normally. The arguments *source* and *destination* determine whether the contents of the source and destination pixrects are NOTted before the *bitblt* operation is performed.

The *combination* argument determines how the source and destination pixrects are combined. If *combination* is equal to REPLACE, the source pixrect (after optionally being NOT-ted) replaces the destination pixrect. If *combination* is equal to AND, OR, or XOR the source pixrect and the destination pixrect are combined in the indicated Boolean fashion. If *combination* is equal to NOT, then the destination is set to a bitwise NOT operation of the source pixrect.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in in state VDOP, VSOP, or VSAC.

Set Global Drawing Mode (SunCGI Extension)

```
Cerror set_global_drawing_mode(combination)
Ccombtype combination; /* combination rules */
```

`set_global_drawing_mode` determines the current *global drawing mode* which in turn determines how *all output primitives except bitblts* are displayed. The *combination* argument determines how the source and destination pixrects are combined. If *combination* is equal to REPLACE (the default value) the output primitive replaces the destination background. If *combination* is equal to AND, OR, or XOR the output primitive and the information on the screen are combined in the indicated Boolean fashion. If *combination* is equal to NOT, then the destination is set to a bitwise NOT operation of the source pixrect.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in in state VDOP, VSOP, or VSAC.

Inquire Drawing Mode

```
Cerror inquire_drawing_mode(visibility, source,
    destination, combination)
Cbmode *visibility; /* transparent or opaque */
Cbitmaptype *source; /* NOT source bits */
Cbitmaptype *destination; /* NOT destination bits */
Ccombtype *combination; /* combination rules */
```

The `inquire_drawing_mode` returns the values of the four components of the current *drawing mode*.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in in state VDOP, VSOP, or VSAC.

4

Attributes

4

Attributes

The current attributes determine how output primitives are displayed. Attributes are *not* specific to any view surface, but affect all view surfaces. The default attributes are defined in Table 4-1. The current attributes may be set either individually or in groups (by changing the index into the *bundle table*). Example programs illustrating these methods of changing attributes are given in Figures 4-1 and 4-2.

Each entry in the *bundle table* specifies a set of attributes for a particular type of primitive (for example, solid objects). The method for setting the current attributes depends on the state of the ASF (*aspect source flag*) for each attribute. For individual attribute functions to have an effect, the ASF must be set to INDIVIDUAL. If the ASF is set to BUNDLED, the current attribute is defined by the entry in the *bundle table* pointed to by the *bundle index*. The actual appearance of objects also depend on the global drawing mode described in Chapter 3.

The majority of this chapter is devoted to individual attribute functions. Individual attribute functions are grouped according to the output primitives they effect: polylines, polymarkers, filled objects, and text. The `color_table` function (which redefines color table entries) is also included in this chapter. Finally, functions for obtaining the values of the current attributes are discussed.

Table 4-1 *Default Attributes*

Attribute	*Value*	*Attribute*	*Value*
All ASF's	INDIVIDUAL	All Bundle Indices	1
Line Color	1	Line Width	0.0
Line Endstyle	BEST_FIT	Line Width Specification Mode	SCALED
Line Type	SOLID		
Marker Color	1	Marker Size	4.0
Marker Size Specification Mode	SCALED	Marker Type	DOT
Fill Color	1	Number of Pattern Table Entries	2
Fill Hatch Index	0		
Fill Pattern Index	1	Pattern Size	300,300
Interior Style	HOLLOW	Pattern Reference Point	0,0
		Pattern with Fill Color	OFF
Perimeter Color	1	Perimeter Width Specification Mode	SCALED
Perimeter Type	SOLID		
Perimeter Width	0.0	Perimeter Visibility	ON
Fontset	1	Text Font	STICK
Fixed Font	0		
Character Base.*x*	1.0	Character Spacing	0.1
Character Base.*y*	0.0	Character Up.*x*	0.0
Character Expansion Factor	1.0	Character Up.*y*	1.0
Character Height	1000	Text Color	1
Character Path	RIGHT	Text Precision	STRING
Horizontal Text Alignment	NRMAL	Text Continuous Alignment.*y*	1.0
Text Continuous Alignment.*x*	1.0	Vertical Text Alignment	NORMAL

4.1. Bundled Attribute Functions

The attribute environment selector functions determine if the current attributes are defined individually or by using a set of attributes (bundles). Bundles are defined by entries in the *bundle table*. The CGI standard specifies the *bundle table* as read-only but **SunCGI** allows user-definition of entries in the *bundle table*. Each type of primitive has its own index into the bundle table, described with its specific attribute functions.

The following example program illustrates how to change the appearance with bundled attributes. The program draws a polyline with a different line style and line width.

```
#include <cgidefs.h>

Ccoor box[5] = { 10000,10000 ,
                 10000,20000 ,
                 20000,20000 ,
                 20000,10000 ,
                 10000,10000 };
Cbunatt bundle = { DASHED_DOTTED, 1., 4,
                   X, 6., 4,
                   PATTERN, 1, 1, 2,
                   DOTTED, 1.5, 1,
                   STICK, CHARACTER,
                   1.3, 0.05, 1 };

main()
{
    Ccoorlist boxlist;
    Cint i, line_bundle = 2, name;
    Cflaglist flags;
    Cvwsurf device;

    boxlist.ptlist = box;
    boxlist.n = 5;
    NORMAL_VWSURF(device, PIXWINDD);

    open_cgi();
    open_vws(&name, &device);

    flags.value = (Casptype *) malloc(18*sizeof(Casptype));
    flags.num = (Cint *) malloc(18*sizeof(Cint));
    for (i = 0; i < 18; i++) {
        flags.value[i] = BUNDLED;
        flags.num[i] = i;
    }
    flags.n = 18;

    define_bundle_index(2, &bundle);
    set_aspect_source_flags(&flags);
    polyline_bundle_index(line_bundle);
    polyline(&boxlist);

    sleep(10);
    close_vws(name);
    close_cgi();
}
```

Figure 4-1 *Example Program with Bundled Attributes*

Set Aspect Source Flags

```
Cerror set_aspect_source_flags(flags)
Cflaglist *flags; /* list of ASFs */
```

`set_aspect_source_flags` determines whether individual attributes are set individually or from bundle table entries.

```
typedef struct {
    Cint n;
    Cint num[];
    Casptype value[];
} Cflaglist;
```

The *n* element of the `flags` argument determines how many flags are to be set. The *num* array of the `flags` argument determines which flags are to be set. Flag numbers are provided in Table 4-2. Finally, the *value* array of the `flags` argument determines the values of the flags specified in *num*. If a value is assigned to INDIVIDUAL, the individual attribute functions affect the current attribute. If the value of index is BUNDLED, calls to individual attribute functions have *no effect*.[12] The default *bundle index* is set to 1 (which initially contains the default value for the attributes specified in Table 4-1). The default value of all *aspect source flags* is INDIVIDUAL.

Errors

`ENOTOPOP` [5] CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.

Table 4-2 *Attribute Source Flag Numbers*

Flag	*Attribute*	*Flag*	*Attribute*
0	line type	9	fill color
1	line width	10	perimeter type
2	line color	11	perimeter width
3	marker type	12	perimeter color
4	marker width	13	text font index
5	marker color	14	text precision
6	interior style	15	character expansion factor
7	hatch index	16	character spacing
8	pattern index	17	text color

Define Bundle Index (SunCGI Extension)

```
Cerror define_bundle_index(index, entry)
Cint index; /* entry in attribute environment table */
Cbunatt *entry; /* new attribute values */
```

`define_bundle_index` defines an entry in the *bundle table*. The type `Cbunatt` is a structure which contains elements corresponding to all the attributes. If the contents of a *bundle table* entry are changed, all subsequently drawn primitives use the information in the new entry, depending on the relevant aspect source flags. You should keep this fact in mind if you are designing display list traversal algorithms using SunCGI.

[12] In fact, SunCGI currently produces error 30 when these individual attribute function is called while the corresponding ASF is BUNDLED.

```
typedef struct {
    Clintype line_type;
    Cfloat line_width;
    Cint line_color;
    Cmartype marker_type;
    Cfloat marker_size;
    Cint marker_color;
    Cintertype interior_style;
    Cint hatch_index;
    Cint pattern_index;
    Cint fill_color;
    Clintype perimeter_type;
    Cfloat perimeter_width;
    Cint perimeter_color;
    Cint text_font;
    Cprectype text_precision;
    Cfloat character_expansion;
    Cfloat character_spacing;
    Cint text_color;
} Cbunatt;
```

In addition to the errors listed below, other errors can be detected if any of the attribute values are invalid, as specified in later sections. Results are undefined if an error occurs.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.
EBBDTBDI [31]	Bundle table index out of range.

4.2. Line Attributes

SunCGI provides for specifying the style, width and color of lines which constitute polylines, circular arcs, and elliptical arcs. The functions do *not* affect the drawing of the perimeter of solid objects which are set by the perimeter functions.

Polyline Bundle Index

```
Cerror polyline_bundle_index(index)
Cint index; /* polyline bundle index */
```

`polyline_bundle_index` sets the current polyline bundle index to the value of *index*. The contents of the *polyline bundle index* are *line type*, *line width* and *line color*. The *line width specification mode* and the *line endstyle* attributes are not included in the polyline bundle. If *index* is not defined, an error is generated, and the `polyline_bundle_index` does not change. If the ASF's for any of these attributes is set to BUNDLED, the current values of these attributes are set to the contents of the bundle.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.
EBADLINX [33]	Polyline index is invalid.

Line Type

```
Cerror line_type(ttyp)
Clintype ttyp; /* style of line */
```

`line_type` defines the line type for polylines. The enumerated type `Clintype` contains values that correspond to valid line types.

```
typedef enum {
    SOLID,
    DOTTED,
    DASHED,
    DASHED_DOTTED,
    DASH_DOT_DOTTED,
    LONG_DASHED
} Clintype;
```

The default line style is SOLID. The actual representation of a line on the screen is affected by the *line endstyle*. DASH_DOT_DOTTED actually has three dots between dashes.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.
EBTBUNDL [30]	ASF is BUNDLED.

Line Endstyle (SunCGI Extension)

```
Cerror line_endstyle(ttyp)
Cendstyle ttyp; /* style of line */
```

`line_endstyle` determines how a textured (non-SOLID) line terminates. The enumerated type `Cendstyle` contains values that correspond to valid line end styles.

```
typedef enum {
    NATURAL,
    POINT,
    BEST_FIT
} Cendstyle;
```

If the endstyle selected is NATURAL, the last component of the line texture (for example, a dash or a dot) which can be completely drawn is drawn. Blank space at the end of the line may cause the line to not appear as long as specified by the starting and ending coordinates. If the endstyle selected is POINT, the last point of the line is drawn whether it is appropriate or not. In this case, the endpoints of the line always appear on the screen. If the endstyle selected is BEST_FIT, the last point is always drawn but is extended as far back as the last space if appropriate. However, the BEST_FIT endstyle may shorten the space between the last element of the line and the element preceding the last element by one in order to guarantee that the line ends on a drawn point. The default endstyle is BEST_FIT.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.

Line Width Specification Mode

```
Cerror line_width_specification_mode(mode)
Cspecmode mode; /* pixels or percent */
```

`line_width_specification_mode` allows the `line_width` to be specified in pixels or as a percentage of VDC space according to the value of mode The enumerated type `Cspecmode` contains values that correspond to line width specification modes.

```
typedef enum {
    ABSOLUTE,
    SCALED
} Cspecmode;
```

If the *line width specification mode* is changed from ABSOLUTE to SCALED, the change in the line width will probably be dramatic. The default *line width specification mode* is SCALED.

If multiple view surfaces are active, the line width is scaled separately for each view surface.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.

Line Width

```
Cerror line_width(index)
Cfloat index; /* line width */
```

`line_width` determines the width of the lines composing polylines, circular arcs, etc. If the *line width specification mode* is SCALED, *index* is expressed in percent of VDC space and if the x and y dimensions are different, the width is calculated on the basis of the range of the x coordinate of VDC space. If the parameter setting would result in a line less than one pixel wide, the line width is displayed as one pixel wide. The default *line width* is 0.0 (SCALED).

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.
EBTBUNDL [30]	ASF is BUNDLED.
EBDWIDTH [34]	Width must be nonnegative.

Line Color

```
Cerror line_color(index)
Cint index; /* line color */
```

`line_color` determines the color of the lines. *index* selects an entry in the color lookup table. The default value of *index* is 1. An error is detected if *index* is not between 0 and 255.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.
EBTBUNDL [30]	ASF is BUNDLED.
ECINDXLZ [35]	Color index is less than zero.

EBADCOLX [36] Color index is invalid.

4.3. Polymarker Attributes

The type, size and color of markers (the components of polymarkers) are controlled by the following functions.

Polymarker Bundle Index

```
Cerror polymarker_bundle_index(index)
Cint index; /* polymarker bundle index */
```

`polymarker_bundle_index` sets the current polymarker bundle index to the value of *index*. The contents of a *polymarker bundle* are *marker type*, *marker size* and *marker color*. The *marker size specification mode* function is not included in the polymarker bundle. If *index* is not defined, an error is generated, and the *polymarker bundle index* does not change. If the ASF's for any of these attributes is set to BUNDLED, the current values of these attributes are set to the values of the corresponding attribute in the bundle.

Errors

ENOTOPOP [5] CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.

EBADMRKX [37] Polymarker index is invalid.

Marker Type

```
Cerror marker_type(ttyp)
Cmartype ttyp; /* style of marker */
```

`marker_type` sets the marker type. The enumerated type `Cmartype` contains values that correspond to valid marker types.

```
typedef enum {
    DOT,
    PLUS,
    ASTERISK,
    CIRCLE,
    X
} Cmartype;
```

Note that all marker types appear as a point when the marker size is very small. The default *marker type* is DOT.

Errors

ENOTOPOP [5] CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.

EBTBUNDL [30] ASF is BUNDLED.

Marker Size Specification Mode

```
Cerror marker_size_specification_mode(mode)
Cspecmode mode; /* pixels or percent */
```

`marker_size_specification_mode` allows the *marker size* to be specified in pixels or as a percentage of VDC space according to the value of *mode*. The enumerated type `Cspecmode` contains values that correspond to valid marker size specifications.

```
typedef enum {
    ABSOLUTE,
    SCALED
} Cspecmode;
```

The default *marker size specification mode* is SCALED.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.

Marker Size

```
Cerror marker_size(index)
Cfloat index; /* marker size */
```

`marker_size` sets the size of the *marker height* and *marker width*. *index* is expressed in percent of VDC space. The default marker size is 4.0 percent of VDC space. If the marker size becomes very small, markers of all types are displayed as points. An error is detected if *index* is negative.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.
EBADSIZE [38]	Size must be nonnegative.

Marker Color

```
Cerror marker_color(index)
Cint index; /* marker color */
```

`marker_color` determines the color of the markers. *index* selects an entry in the color lookup table. An error is detected if *index* is not between 0 and 255. The default *marker color* is 1.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.
EBTBUNDL [30]	ASF is BUNDLED.
ECINDXLZ [35]	Color index is less than zero.
EBADCOLX [36]	Color index is invalid.

4.4. Solid Object Attributes

The solid object attribute functions describe how all solid object primitives are filled (colored-in). There are three sets of solid object attribute functions:

fill area attributes
: The fill area attribute functions determine the general method for filling solid geometrical objects.

hatch and pattern attributes
: determines a pixel array for filling a polygon if the *fill style* is set to PATTERN.

perimeter attributes
: determine how the boundary of a geometrical object is displayed if the *perimeter visibility* is ON.

Fill Area Bundle Index

```
Cerror fill_area_bundle_index(index)
Cint index; /* fill area bundle index */
```

`fill_area_bundle_index` sets the current *fill area bundle index* to the value of *index*. The contents of the *fill area bundle* are *interior style*, *fill color hatch index pattern index perimeter type perimeter width* and *perimeter color*. The *perimeter width specification mode* and the pattern attributes are not included in the definition of the fill area bundle. If *index* is not defined, an error is generated, and the fill area bundle index does not change. If the ASF's for any of these attributes is set to BUNDLED, the current value of the attribute is set to the value of the corresponding attribute in the bundle.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.
EBADFABX [39]	Fill area index is invalid.

Interior Style

```
Cerror interior_style(istyle, perimvis)
Cintertype istyle; /* fill style */
Cflag perimvis; /* perimeter visibility */
```

`interior_style` sets the *fill style* for solid objects. The enumerated type `Clintertype` contains values that correspond to valid line types.

```
typedef enum {
    HOLLOW,
    SOLIDI,
    PATTERN,
    HATCH
} Cintertype;
```

If the *fill style* is set to SOLIDI, the solid object is filled with the current *fill color*. If *istyle* is set to PATTERN or HATCH, the solid object is filled with the current PATTERN or HATCH style. The PATTERN and HATCH styles are explained in the pattern attributes section. The default *fill style* is HOLLOW.

`interior_style` also determines whether the perimeter of the solid object is visible according to the value of *perimvis* (which must be ON or OFF). If *perimvis* is OFF, the perimeter attributes have no effect. The default value of *perimeter visibility* is ON.

Be careful when using the *interior style* function to explicitly specify the *perimvis* argument. If you do not specify it, or set it to OFF, the geometrical output primitive may not be displayed because the *interior style* is HOLLOW.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.

4.5. Solid Interior Fill Attribute

The following section contains the description of a function that determines the color of an interior region if the *fill style* is not HOLLOW.

Fill Color

```
Cerror fill_color(color)
Cint color; /* color for solid object fill */
```

`fill_color` determines the color for filling solid objects, if the *fill style* is not set to HOLLOW.

The default *fill style* is HOLLOW, so changing the *fill color* will not have an effect without changing the *interior style* first. The default *fill color* is 1. An error is detected if *fill color* is not between 0 and 255.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.
ECINDXLZ [35]	Color index is less than zero.
EBADCOLX [36]	Color index is invalid.

4.6. Hatch and Pattern Attributes

Geometrical primitives can be filled with 2D arrays of color values called patterns. SunCGI supports pre-defined as well as user-defined patterns. The definition of patterns is stored in the *pattern table*. Each entry in the pattern table consists of a 2D array of color values and the x and y dimensions of the array. The starting position (upper left-hand corner) of the pattern is determined by the *pattern reference point*.

Two types of patterns are available: PATTERNs and HATCHes. PATTERNs can be scaled and translated. HATCHes can't and simply fill the geometrical output primitives with pixel arrays.

The following example program illustrates how to change the appearance with the individual attribute functions. The program draws a polygon and fills it with a pattern.

```
#include <cgidefs.h>

Ccoor box[5] = { 10000,10000 ,
                 10000,20000 ,
                 20000,20000 ,
                 20000,10000 ,
                 10000,10000 };
Cint pattern[16] = { 50, 75, 100, 125,
                     150, 0, 0, 175,
                     200, 0, 0, 225,
                     250, 275, 300, 325 };

main()
{
    Ccoorlist boxlist;
    Cint dx = 250, dy = 250, index = 2, name;
    Cvwsurf device;

    boxlist.n = 5;
    boxlist.ptlist = box;
    NORMAL_VWSURF(device, PIXWINDD);

    open_cgi();
    open_vws(&name, &device);

    interior_style(PATTERN, ON);
    pattern_table(index, 4, 4, pattern);
    pattern_index(index);
    pattern_size(dx, dy);
    polygon(&boxlist);

    sleep(10);

    close_vws(name);
    close_cgi();
}
```

Figure 4-2 *Example Program with Bundled Attributes*

Hatch Index

```
Cerror hatch_index(index)
Cint index; /* HATCH index in the pattern table */
```

`hatch_index` determines which entry in the pattern table is used to fill solid objects when the *fill style* is set to HATCH. The default *hatch index* is 0. An error is generated if *index* points to an undefined entry in the pattern table.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.

EBTBUNDL [30]	ASF is BUNDLED.
ESTYLLEZ [42]	Style (pattern or hatch) index is less than zero.
ENOPATNX [43]	Pattern table index not defined.

Pattern Index

```
Cerror pattern_index(index)
Cint index; /* PATTERN index in the pattern table */
```

`pattern_index` determines which index in the pattern table is used to fill solid objects when the *fill style* is set to PATTERN. The default *pattern index* is 1. An error is generated if *index* points to an undefined entry in the pattern table.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.
EBTBUNDL [30]	ASF is BUNDLED.
ESTYLLEZ [42]	Style (pattern or hatch) index is less than zero.
ENOPATNX [43]	Pattern table index not defined.

Pattern Table

```
Cerror pattern_table(index, m, n, colorind)
Cint index; /* entry in table */
Cint m, n; /* number of rows and columns */
Cint *colorind; /* array containing pattern */
```

`pattern_table` defines an entry in the pattern table. *index* defines the entry in the table (which must be less than 50). An error is generated if *index* is outside the bounds of the *pattern table*. *m* and *n* define the height and width of the pattern (in pixels). The array pointed to by the argument *colorind* contains the actual pattern row-wise from the upper left. For monochrome view surfaces, all nonzero entries in `colorind` are treated as 1 when used. The maximum number of elements in a pattern ($m \times n$) is MAXPATSIZE.

Pattern 0 is initially defined to be a 3×3 matrix which is set to zero at the corners and one elsewhere. Pattern 0 produces simple cross-hatching. Pattern 1 (which produces a polka-dot pattern) is initially defined to be a 3×3 matrix which is set to 1 at the center and 0 elsewhere.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.
EPATARTL [40]	Pattern array too large.
EPATSZTS [41]	Pattern size too small.
ESTYLLEZ [42]	Style (pattern or hatch) index is less than zero.
EPATITOL [44]	Pattern table index too large.

Pattern Reference Point

```
Cerror pattern_reference_point(begin)
Ccoor *begin;
```

`pattern_reference_point` defines the point in VDC space where the

pattern box begins. The pattern is then replicated over all VDC space. The upper left-hand corner of the *pattern box* is determined by *begin*. The default *pattern reference point* is (0, 0). `pattern_reference_point` has no effect if the *interior style* is not set to PATTERN.

Errors

ENOTOPOP [5] CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.

Pattern Size

```
Cerror pattern_size(dx, dy)
Cint dx, dy; /* size of pattern in VDC space */
```

`pattern_size` defines the size of the pattern array in VDC coordinates. *dx* and *dy* determine the size of an element of the pattern in VDC space. `pattern_size` therefore allows you to 'stretch' the pattern to a certain size. If *dx* or *dy* would result in pattern elements less than one pixel wide, 1 is used. If the *pattern size* is larger than the bounds of screen space, the effective *pattern size* is the size of VDC space. The default *pattern size* is (300, 300).

Errors

ENOTOPOP [5] CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.

Pattern with Fill Color (SunCGI Extension)

```
Cerror pattern_with_fill_color(flag)
Cflag flag; /* ON to use nonzero pattern
    elements as fill color */
```

Binary patterns allow the same pattern to be applied in different colors, without redefining the pattern array. `pattern_with_fill_color` sets a non-standard CGI state *pattern with fill color*. The default *pattern with fill color* is OFF and each color value in a pattern table entry is used verbatim, as in standard CGI. When a pattern is used while *flag* is ON, the pattern is considered to be a 2D array of flags: where the pattern element is nonzero, the current fill color is used, instead of the actual value of the pattern element. (When *pattern with fill color* is zero, a zero color index is used, just as when the flag is OFF.)

4.7. Perimeter Attributes

The following sections contain descriptions of functions that determine the perimeter attributes *perimeter type*, *perimeter width*, *perimeter width specification mode* and *perimeter color*.

Perimeter Type

```
Cerror perimeter_type(ttyp)
Clintype ttyp; /* style of perimeter */
```

`perimeter_type` defines the perimeter type for solid objects. The enumerated type `Clintype` contains values that correspond to valid perimeter types.

```
typedef enum {
    SOLID,
    DOTTED,
    DASHED,
    DASHED_DOTTED,
    DASH_DOT_DOTTED,
    LONG_DASHED
} Clintype;
```

The default perimeter style is SOLID. Notice that there is no ending style for perimeter. The endstyle is controlled by the `line_endstyle` function.

As mentioned previously, control of the drawing of the borders of solid objects is under the control of the perimeter attribute functions, not the line attribute functions. However, the two sets of functions take the same values. The perimeter attributes are essentially the same as the line attributes except that they affect the borders of solid attributes. The appearance of a perimeter can be similar to a line especially if *interior style* is set to HOLLOW. Perimeter attribute functions have no effect if the *perimeter visibility* is set to OFF.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.
EBTBUNDL [30]	ASF is BUNDLED.

Perimeter Width

```
Cerror perimeter_width(width)
Cfloat width; /* perimeter width */
```

`perimeter_width` determines the width of the perimeters of solid objects. *index* can be expressed in percent of VDC space or pixels. If the *perimeter width specification mode* is set to SCALED and the x and y dimensions are different, the *perimeter width* is calculated on the basis of the range of the x coordinate of VDC space. If the parameter setting would result in a perimeter less than one pixel wide, the perimeter width is displayed as one pixel wide. The default *perimeter width* is 0.0 (SCALED).

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.
EBTBUNDL [30]	ASF is BUNDLED.
EBDWIDTH [34]	Width must be nonnegative.

Perimeter Width Specification Mode

```
Cerror perimeter_width_specification_mode(mode)
Cspecmode mode; /* pixels or percent */
```

`perimeter_width_specification_mode` allows the `perimeter_width` to be specified in pixels or as a percentage of VDC space according to the value of mode (which can either be ABSOLUTE or SCALED). If the *perimeter width specification mode* is changed from ABSOLUTE to SCALED, the change in the line width will probably be dramatic. The default *perimeter width specification mode* is SCALED.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.

Perimeter Color

```
Cerror perimeter_color(index)
Cint index; /* perimeter color */
```

`perimeter_color` determines the color of the perimeters. *index* selects an entry in the color lookup table. The default value of *index* is 1. An error is detected if *index* is not between 0 and 255.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.
EBTBUNDL [30]	ASF is BUNDLED.
ECINDXLZ [35]	Color index is less than zero.
EBADCOLX [36]	Color index is invalid.

4.8. Text Attributes

SunCGI provides a variety of functions for determining how text is written to the screen. The most important text attribute is *text precision*. If *text precision* is set to STRING, firmware characters are used. The fonts, size, spacing, and alignment of firmware are more limited than characters drawn with *text precision* set to a value other than STRING. Therefore, calls to text attribute functions regulating these aspects of text drawing have no effect when *text precision* is set to STRING.

Text Bundle Index

```
Cerror text_bundle_index(index)
Cint index; /* text bundle index */
```

`text_bundle_index` sets the current *text bundle index* to the value of *index*. The contents of the *text bundle index* are *text font text precision*, *character expansion factor*, *character spacing*, and *text color*. The *character height character orientation character path text alignment* and *fixed font* are *not* included in the definition of the text bundle. If *index* is not defined, an error is generated, and the *text bundle index* does not change. If the ASF's for any of these attributes are set to BUNDLED, the current values of these attributes are set to the contents of the bundle.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.
EBADTXTX [45]	Text index is invalid.

Text Precision

```
Cerror text_precision(ttyp)
Cprectype ttyp; /* text type */
```

`text_precision` controls the precision with which text is displayed. The enumerated type `Cprectype` contains values that correspond to valid text precisions.

```
typedef enum {
    STRING,
    CHARACTER,
    STROKE
} Cprectype;
```

If the *text precision* is set to STRING, the firmware character set is used. Note: firmware characters cannot be scaled or rotated.

Characters are clipped, but not in parts (that is, if any portion of the character exceeds the clipping boundary the whole character is clipped). If the *text precision* is set to CHARACTER, software generated characters are employed and characters are clipped, but not in parts. All text attributes have a visible effect on software generated characters. If the *text precision* is set to STROKE, the CHARACTER precision capabilities are enabled and characters are clipped in parts. The default *text precision* is STRING.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.
EBTBUNDL [30]	ASF is BUNDLED.

Character Set Index

```
Cerror character_set_index(index)
Cint index; /* font set */
```

`character_set_index` selects a set of fonts. Although SunCGI supports this function, only set number 1 is defined. Calls to `character_set_index` with *index* assigned to a value other than 1 are ignored.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.

Text Font Index

```
Cerror text_font_index(index)
Cint index; /* font */
```

`text_font_index` determines the current font. A list of available fonts and their availability when *text precision* is set to STRING is given in Table 4-3. A warning about the SYMBOL font: undefined characters are displayed as bugs (the six-legged kind). The default font is STICK.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.
EBTBUNDL [30]	ASF is BUNDLED.
ETXTFLIN [47]	Text font is invalid.

Table 4-3 *Available Fonts*

Font	*String Precision*
ROMAN	Yes
GREEK	Yes†
SCRIPT	Yes
OLDENGLISH	No
STICK	Yes
SYMBOLS	No

† displayed as STICK font.

Character Expansion Factor

```
Cerror character_expansion_factor(efac)
Cfloat efac; /* width factor */
```

`character_expansion_factor` determines the width-to-height ratio of characters. If *efac* is greater than 1 the characters appear fatter than they are wide. If *efac* is less than 1 the characters appear slimmer than they are wide. The default *character expansion factor* is 1.0. An error is generated if *efac* is less than 0.01 or greater than 10.

Errors

`ENOTOPOP` [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.
`EBTBUNDL` [30]	ASF is BUNDLED.
`ECEXFOOR` [48]	Expansion factor is out of range.

Character Spacing

```
Cerror character_spacing(spcratio)
Cfloat spcratio; /* spacing ratio */
```

`character_spacing` sets the spacing between characters based on the height of the characters. The amount of space between characters is obtained by multiplying the character height by *spcratio*. The default *character spacing factor* is 0.1. An error is generated if *spcratio* is less than -10 or greater than 10.

Errors

`ENOTOPOP` [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.
`EBTBUNDL` [30]	ASF is BUNDLED.
`ECEXFOOR` [48]	Expansion factor is out of range.

Character Height

```
Cerror character_height(height)
Cint height; /* height in VDC */
```

The `character_height` function determines the height of text in VDC units. The height is defined as the distance from the top to the bottom of the character.

Notice that changing the character height implicitly changes the *character spacing*.

The default character height is 1000. This may result in huge characters if VDC space is reset from its default range (0-32767). If the x and y dimensions of VDC space are different, the height is calculated on the basis of the range of the x coordinate of VDC space.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.
EBTBUNDL [30]	ASF is BUNDLED.
ECHHTLEZ [49]	Character height is less than or equal to zero.

Fixed Font (SunCGI Extension)

```
Cerror fixed_font(flag)
Cint flag; /* fixed or variable width characters */
```

`fixed_font` allows characters to be of fixed or variable size. If *flag* is nonzero, the characters are of uniform size, otherwise the characters are packed proportional to their actual sizes. If the *character precision* is STRING, this function has no effect. By default SunCGI supports variable width characters.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.

Text Color

```
Cerror text_color(index)
Cint index; /* color */
```

`text_color` determines the color of the text. *index* selects an entry in the color lookup table. The default value of *index* is 1. An error is detected if *index* is not between 0 and 255.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.
EBTBUNDL [30]	ASF is BUNDLED.
ECINDXLZ [35]	Color index is less than zero.
EBADCOLX [36]	Color index is invalid.

Character Orientation

```
Cerror character_orientation(xbase, ybase, xup, yup)
Cfloat xbase, ybase, xup, yup;
    /* character base and up vectors */
```

`character_orientation` specifies the skew and direction of text. The left side of the character box lies on an invisible line called the *character up vector* whose slope is determined by *xup* and *yup*. The bottom of the character box lies on an invisible line called the *character base vector* whose slope is determined by *xbase* and *ybase*.

If the *character up vector* and the *character base vector* are not orthogonal, the text is distorted. Calls to `character_orientation` have no effect if *text precision* is set to STRING. The default values for the *character up vector* and the *character base vector* are *xbase* = 1.0, *ybase* = 0.0, *xup* = 0.0, and *yup* =

1.0.

The *character up vector* and the *character base vector* influence the *character path* and the character alignment. For example, if *xbase* = -1.0 and the character path is RIGHT, the text is written to the *left*.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.
ECHRUPVZ [50]	Length of character up vector or character base vector is zero.

Character Path

```
Cerror character_path(path)
Cpathtype path; /* text direction */
```

`character_path` specifies the direction in which text is written. The enumerated type `Cpathtype` contains values that correspond to valid character paths.

```
typedef enum {
    RIGHT,
    LEFT,
    UP,
    DOWN
} Cpathtype;
```

The actual effect of `character_path` depends on the *character up vector* and the *character base vector*. RIGHT specifies that the text is written in the direction of the *character base vector*. For example, if the direction of the *character base vector* points left instead of right (*xup* = -1.0 instead of 1.0), the text will be written right-to-left instead of left-to-right which is the usual interpretation of RIGHT. LEFT specifies that the text is written in the opposite direction of the *character base vector*. The *character up vector* and *character base vector* essentially change functions when the character direction is set to UP or DOWN. UP specifies that the text is written in the direction of the *character up vector*. DOWN specifies that the text is written in the opposite direction of the *character up vector*. The default *character path* is RIGHT.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.

Text Alignment

```
Cerror text_alignment(halign, valign, hcalind, vcalind)
Chaligntype halign; /* horizontal alignment type */
Cvaligntype valign; /* vertical alignment type */
Cfloat hcalind, vcalind;
    /* continuous alignment indicators */
```

`text_alignment` determines where the text is positioned relative to the starting point specified by the *c1* argument of the `text` or `vdm_text` function. *halign* determines where the character is placed in relation to the *x* component of the starting coordinate of the text position (specified by the *c1* argument of *text*). The enumerated type `Chaligntype` contains values that correspond to

valid horizontal alignments.

```
typedef enum {
    LFT,
    CNTER,
    RGHT,
    NRMAL,
    CNT
} Chaligntype;
```

If the value of *halign* is LFT, the horizontal position of the text will begin at the left edge of the box enclosing the text. Similarly, if the value of *halign* is RGHT, the horizontal position of the text will begin at the right edge of the box enclosing the text. If the value of *halign* is CNTER the horizontal position of the text will begin equidistant from the right and the left edges of the text box. NRMAL assigns the alignment based on the value of the *character path* (see Table 4-4). If the value of *halign* is CNT (continuous) the horizontal position of the text is determined by the argument *hcalind*. In this case, the text will begin *hcalind* fraction of the width of the text box from the left edge of the character box. The default value of *halign* is NRMAL.

valign specifies where the character is placed in relation to the *y* component of the text position. The enumerated type `Cvaligntype` contains values that correspond to valid vertical alignments.

```
typedef enum {
    TOP,
    CAP,
    HALF,
    BASE,
    BOTTOM,
    NORMAL,
    CONT
} Cvaligntype;
```

If the value of *valign* is TOP, the vertical position of the text will begin at the top edge of the character box. If the value of *valign* is CAP, the vertical position of the text will begin at the *cap line* of the character.[13] Similarly, if the value of *valign* is BOTTOM, the vertical position of the text will begin at the bottom edge of the character box. If the value of *valign* is BASE, the vertical position of the text will begin at the *baseline* of the character.[14] If the value of *valign* is HALF the vertical position of the text will begin equidistant from the top and the bottom edges of the character box. NORMAL assigns the alignment based on the value of the *character path* (see Table 4-4). If the value of *valign* is assigned to CONT (continuous), the vertical position of the text is determined by the argument *vcalind* and will begin *vcalind* fraction of the height of the character box from the bottom edge of the character box. The default value of *valign* is NORMAL.

[13] The *cap line* is defined as the invisible line corresponding to the top of the average character within a font.

[14] The *baseline* is defined as the invisible line corresponding to the bottom of the average character within a font. The *baseline* does not necessarily correspond to the bottom of a character. For example, a the tail of a lower-case g extends below the baseline.

Table 4-4 *Normal Alignment Values*

Character Path	*Horizontal Normal*	*Vertical Normal*
RIGHT	LEFT	BASELINE
LEFT	RIGHT	BASELINE
UP	CENTER	BASELINE
DOWN	CENTER	TOP

Errors

ENOTOPOP [5] CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.

4.9. Color Attributes

SunCGI supports only one color specification mode — INDEXED. This color specification mode means that the red, green, and blue values (hereafter referred to as RGB values) are obtained from a table known as the *color lookup table*. The initial values of the *color lookup table* are provided in Table 4-5. If the device is monochrome, nonzero color values are displayed as black; zero is displayed as white.

Table 4-5 *Default Color Lookup Table*

Index	*Color*
0	black
1	red
2	yellow
3	green
4	cyan
5	blue
6	magenta
7	white

Color Table

```
Cerror color_table(istart, clist)
Cint istart; /* starting address */
Ccentry *clist; /* color triples and number of entries */
```

`color_table` defines RGB entries into the *color lookup table*. The color lookup table is initialized based on the depth of the display frame buffer and the *cmapsize* field provided in the `Cvwsurf` structure provided to `open_vws`. A monochrome device has an unwritable color map; non-zero color indices are displayed as black, zero is displayed as white. A color device gets a color map segment with 8 entries if the cmapsize field is zero upon opening the view surface. The 8 default color values are given in Table 4-5. Larger color maps are also initialized to evenly spaced RGB values.

The structure `Ccentry` contains elements that describe a color map entry.

```
typedef struct {
    unsigned char *ra;
    unsigned char *ga;
    unsigned char *ba;
    Cint n;
} Ccentry;
```

The minimum and maximum color table entries are treated specially by Pixwins and hence by SunCGI. If they are set to be the same value, the user's values for these two entries are *both* ignored. They revert to the inverse of the normal values; entry 0 becomes white, the maximum entry becomes black.

The argument *istart* determines the first entry in the color lookup table to be modified. the argument *clist* contains the color information for entry *istart* in terms of triples of values of numbers ranging between 0 and 255. The last field of *clist* reports how many entries are to be modified. An error is generated if either the indices to the *color lookup table* are out of range.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.
ECINDXLZ [35]	Color index is less than zero.
EBADCOLX [36]	Color index is invalid.

4.10. Inquiry Functions

The attribute inquiry functions permit examination of the current attributes. Attributes are reported in groups corresponding to the class of output primitive which they modify. The argument to each inquiry function has its own structure type which has an element for each of the individual attributes (see Appendix D).

Inquire Line Attributes

```
Clinatt *inquire_line_attributes()
    /* returns a pointer to line attribute structure */
```

`inquire_line_attributes` reports the current *line style*, *line width*, *line color*, and *polyline bundle index* in the appropriate elements of the returned value of the function.

```
typedef struct {
    Clintype style;
    Cfloat width;
    Cint color;
    Cint index;
} Clinatt;
```

`inquire_line_attributes` returns a NULL (not an error number) in case of errors. Errors are printed if the error warning mode is not set to NO_ACTION.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.

Inquire Marker Attributes

```
Cmarkatt *inquire_marker_attributes()
    /* returns a pointer to marker attribute structure */
```

`inquire_marker_attributes` reports the current *marker style*, *marker width*, *marker color*, and *polymarker bundle index* in the appropriate elements of the returned value of the function.

```
typedef struct {
    Cmartype type;
    Cfloat size;
    Cint color;
    Cint index;
} Cmarkatt;
```

`inquire_marker_attributes` returns a NULL (not an error number) in case of errors. Errors are printed if the error warning mode is not set to NO_ACTION.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.

Inquire Fill Area Attributes

```
Cfillatt *inquire_fill_area_attributes()
```

The current *interior style*, *perimeter visibility*, *fill color*, *hatch index*, *pattern index*, *fill area bundle index*, *perimeter style*, *perimeter width*, and *perimeter color* can be obtained by using the `inquire_fill_attributes` function.

```
typedef struct {
    Cintertype style;
    Cflagtype visible;
    Cint color;
    Cint hatch_index;
    Cint pattern_index;
    Cint index;
    Clintype pstyle;
    Cfloat pwidth;
    Cint pcolor;
} fillatt;
```

`inquire_fill_area_attributes` returns a NULL (not an error number) in case of errors. Errors are printed if the error warning mode is not set to NO_ACTION.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.

Inquire Pattern Attributes

```
Cpatternatt *inquire_pattern_attributes()
    /* returns a pointer to pattern attribute structure */
```

`inquire_pattern_attributes` reports the *current pattern index*, *row count*, *column count*, *color list*, *pattern reference point*, and *pattern size*.

```
typedef struct {
    Cint cur_index;
    Cint row;
    Cint column;
    Cint *colorlist;
    Ccoor *point;
    Cint dx;
    Cint dy;
} patternatt;
```

`inquire_pattern_attributes` returns a NULL (not an error number) in case of errors. Errors are printed if the error warning mode is not set to NO_ACTION.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.

Inquire Text Attributes

```
Ctextatt *inquire_text_attributes()
    /* returns a pointer to text attribute structure */
```

`inquire_text_attributes` reports the current *font set*, *text bundle index*, *font*, *text precision*, *character expansion factor*, *character spacing*, *text color*, *character height*, *character base vector*, *character up vector*, *character path*, and *text alignment*.

```
typedef struct {
    Cint fontset;
    Cint index;
    Cint current_font;
    Cprectype precision;
    Cfloat exp_factor;
    Cfloat space;
    Cint color;
    Cint height;
    Cfloat basex;
    Cfloat basey;
    Cfloat upx;
    Cfloat upy;
    Cpathtype path;
    Chaligntype halign;
    Cvaligntype valign;
    Cfloat hcalind;
    Cfloat vcalind;
} textatt;
```

`inquire_text_attributes` returns a NULL (not an error number) in case of errors. Errors are printed if the error warning mode is not set to NO_ACTION.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.

Inquire Aspect Source Flags

```
Cflaglist *inquire_aspect_source_flags()
    /* returns a pointer to text attribute structure */
```

`inquire_aspect_source_flags` reports whether attributes are set individually by returning all of the values of the ASFs. The element *n* of the flaglist struct is set to 18. The definitions of each flag are in Table 4-2.

```
typedef struct {
    Cint n;
    Cint *num;
    Casptype *value;
} Cflaglist;
```

`inquire_aspect_source_flags` returns a NULL (not an error number) in case of errors. Errors are printed if the error warning mode is not set to NO_ACTION.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.

5

Input

5

Input

CGI has a collection of functions for managing input devices. The design of these functions has two purposes: provide an interface close to the actual input device and maintain portability of applications. CGI accomplishes the first goal with different input device classes and methods of extracting input values. The second goal is achieved through CGI's model of logical input devices (LID), an abstraction whereby logical input devices required by the CGI standard are mapped onto the physical devices available to a CGI implementation. This section will introduce some of the terms used in describing the functionality of the CGI input primitives.

A CGI input device consists of a *measure* associated with a *trigger*. A *measure* is the current value of a logical input device. For example, the IC_LOCATOR device reports an *x-y* position. This device is useful for determining a position on the screen. A *trigger* is a physical device used by an operator to accept a *current value*. A *trigger fire* corresponds to an event on a physical input device. At the request of the application program, **SunCGI** associates a measure with a trigger. Table 5-1 has a list of the five logical input devices available to **SunCGI** application programs and the available triggers. For example, a mouse button on a Sun workstation is a trigger that can be associated with a IC_LOCATOR device. When the mouse button is pressed, the *x-y* position of the mouse is returned as the measure of the IC_LOCATOR input device.

An *input event* is the information saved when a trigger fires. This includes the measure of a logical input device associated with a trigger.

Table 5-1 *Input Devices Offered by* SunCGI

Device Class	*Measure*	*Trigger Number*	*Trigger*
IC_LOCATOR	*x-y* position in VDC space.	2	Left mouse button
		3	Middle mouse button
		4	Right mouse button
		5	Mouse movement†
		6	Mouse still‡
IC_STROKE	Array of *x-y* points in VDC space.	2	Left mouse button
		3	Middle mouse button
		4	Right mouse button
IC_VALUATOR	Normalized *x* position.	2	Left mouse button
		3	Middle mouse button
		4	Right mouse button
		5	Mouse movement
		6	Mouse still
IC_CHOICE	A non-negative integer which represents a selection from a number of choices. Zero represents "no choice".	2	Left mouse button
		3	Middle mouse button
		4	Right mouse button
IC_STRING	Character string.	1	Keyboard input terminated a carriage return.

† The *Mouse Movement* trigger fires when the mouse moves.

‡ The *Mouse Still* trigger fires when the mouse does not move for one fifth of a second or more.

The graphical method with which the measure of an input device is displayed is called *tracking*. SunCGI provides several methods of tracking for each input device. Table 5-3 has a list of track types available for each input device class. Tracking must be explicitly enabled for each device.

Each input device can be in one of the five states described pictorially in Figure 5-1. The state of an input device determines the manner in which the application program retrieves the measure of the input device. The input functions that allow a change of state are listed next to the arrows indicating the state change.

RELEASED

Before an input device is initialized it is in the RELEASED state. Any input function (except initialization) will generate an error in this state.

NO_EVENTS

After an input device has been initialized it is in the NO_EVENTS state. An application program can extract an input value of an input device in NO_EVENTS state. This will result in either the value that the device was

initialized with or the value the device had when it was in a state where it could process events. This is not necessarily the *current* measure of the device and does not change while the device is in this state.

RESPOND_EVENT

The RESPOND_EVENT state corresponds with synchronous communication between the process that controls the input device and the application program. When an application program requests the measure of an input device in RESPOND_EVENT state, SunCGI blocks program execution until it can fulfill the request. The `request_input` function will return when the trigger fires and the input request is satisfied or after a timeout period. The input device then reverts to NO_EVENTS state.

The function that requests input and puts the input device in RESPOND_EVENT state is `request_input`. When the trigger associated with an input device in RESPOND_EVENT state fires, the measure of that input device is then stored in the request register as well as returned by the `request_input` function.

REQUEST_EVENT

The REQUEST_EVENT state corresponds with asynchronous communication between the process that controls the input device and the application program. When an application samples an input device, input handling and program execution continue in parallel. Either the requested trigger fires or an explicit request is made to disable event processing and return the device to NO_EVENTS state.

When the trigger associated with an input device in REQUEST_EVENT state fires, the measure of that input device is then stored in the *request register*, a buffer with one element per device. The request register can be then be read with `get_last_requested_event`.

QUEUE_EVENT

When a device is in QUEUE_EVENT mode, events associated with the indicated device are appended to the *event queue*, a first-in, first-out (FIFO) buffer shared by *all* input devices. After calling `enable_events`, the SunCGI application retains program control. While an input device is in QUEUE_EVENT mode, events are simultaneously added to the event queue when the program executes.

`await_event` returns the event at the head of the event queue. If the queue is empty, `await_event` will wait for the designated trigger to fire or a timeout. The application program must process this queue in a timely fashion or it will *overflow*. The event queue can be flushed completely or for a specific device. The application program must make an explicit request to disable event queue processing and return an input device to NO_EVENTS state.

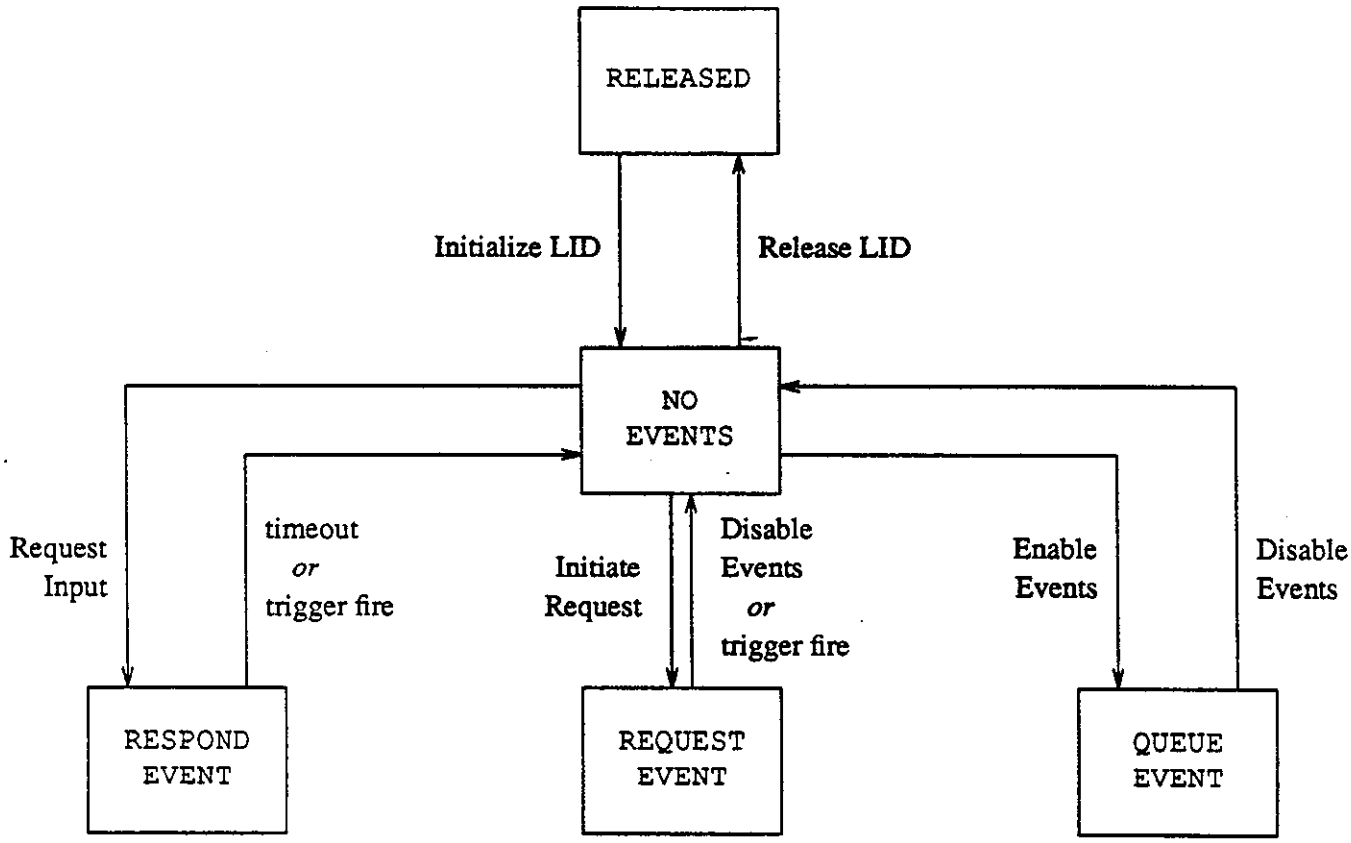

Figure 5-1 *CGI Input State Model*

5.1. Input Device Initialization

Before input can be processed, an input devices must be initialized and associated with a trigger. Input device initialization requires at least one active view surface. Typically, the procedure for initializing an input device includes calls to the `initialize_lid` and `associate` functions which turn on an input device and associate it with a specific trigger.

Initialize LID

```
Cerror initialize_lid(devclass, devnum, ival)
Cdevoff devclass; /* device type */
Cint devnum; /* device number */
Cinrep *ival; /* initial value of device measure */
```

`initialize_lid` initializes an input device and changes its state from RELEASED to NO_EVENTS. This function must be called for an input device before it can be referenced by any other input function. The argument *devclass* specifies the desired type of input value. *devnum* indicates the number of the device within that class. The argument *ival* sets the initial measure of the device.

The `Cinrep` structure contains different elements for each type of measure. The appropriate element of `Cinrep` must be set or an error will be generated.

```
typedef struct {
    Ccoor *xypt; /* LOCATOR */
    Ccoorlist *points; /* STROKE devices */
    Cfloat val; /* VALUATOR device */
    Cint choice; /* CHOICE devices */
    Cchar *string; /* STRING device */
    Cpick *pick; /* PICK devices (unsupported) */
} Cinrep;
```

For example, in a LOCATOR device initialization, the *xypt* field of `Cinrep` must be set to the address of a `Ccoor` allocated by the application program before the *x* and *y* elements can be set. See the example program in Figure 5-2.

Notice that whenever a device is initialized, no associations with triggers are made. This must be done by having the application program call the appropriate functions. An error is generated by `initialize_lid` if the device does not exist, if it is already initialized, or if the initial value is out of range.

Errors

`ENOTVSAC` [4]	CGI not in proper state: CGI shall be in state VSAC.
`EINDNOEX` [80]	Input device does not exist.
`EINDALIN` [82]	Input device already initialized.[15]
`EBADDATA` [95]	Contents of input data record are invalid.
`ESTRSIZE` [96]	Length of initial string is greater than the implementation defined maximum.

Release Input Device

```
Cerror release_input_device(devclass, devnum)
Cdevoff devclass; /* device type */
Cint devnum; /* device number */
```

`release_input_device` releases all associations between a device and its triggers, and removes all pending events for the device from the event queue. `release_input_device` changes the state of the specified input device from NO_EVENTS to RELEASED. An error is produced if *devclass* and *devnum* does not refer to an existing and initialized device.

Errors

`ENOTVSAC` [4]	CGI not in proper state: CGI shall be in state VSAC.
`EINDNOEX` [80]	Input device does not exist.
`EINDINIT` [81]	Input device not initialized.

Associate

```
Cerror associate(trigger, devclass, devnum)
Cint trigger; /* trigger number */
Cdevoff devclass; /* device type */
Cint devnum; /* device number */
```

[15] The ANSI standard allows initialized input devices to be re-initialized. SunCGI does not because it is felt that re-initialization is usually a mistake.

`associate` links a trigger with a specific device. The trigger numbers available for each device are listed in Table 5-1. Multiple associations are allowed; however, some associations are not allowed (for example, IC_LOCATOR may not be associated with the keyboard).

The interaction between an IC_STROKE device and the trigger requires some additional explanation. IC_STROKE can only be associated with the mouse buttons. The first coordinate in the IC_STROKE array is entered when the mouse button is initially pressed, the last coordinate is entered when the mouse button is released. For IC_LOCATOR and IC_VALUATOR devices, the measure is reported when the mouse button is pressed.

Errors

ENOTVSAC [4]	CGI not in proper state: CGI shall be in state VSAC.
EINDNOEX [80]	Input device does not exist.
EINDINIT [81]	Input device not initialized.
EINASAEX [83]	Association already exists.
EINAIIMP [84]	Association is impossible.
EINTRNEX [86]	Trigger does not exist.

Set Default Trigger Associations

```
Cerror set_default_trigger_associations(devclass, devnum)
Cdevoff devclass; /* device type */
Cint devnum; /* device number */
```

`set_default_trigger_associations` associates a device with a default trigger. The default associations are listed in Table 5-2. The rules for trigger association are the same as those for the `associate` function.

Table 5-2 *Default Trigger Associations*

Device Class	*Trigger Number*	*Trigger*
IC_LOCATOR	5	Mouse position
IC_STROKE	4	Right mouse button
IC_VALUATOR	3	Middle mouse button
IC_CHOICE	2	Left mouse button
IC_STRING	1	Keyboard

Errors

ENOTVSAC [4]	CGI not in proper state: CGI shall be in state VSAC.
EINDNOEX [80]	Input device does not exist.
EINDINIT [81]	Input device not initialized.
EINASAEX [83]	Association already exists.
EINTRNEX [86]	Trigger does not exist.

Dissociate

```
Cerror dissociate(trigger, devclass, devnum)
Cint trigger; /* trigger number */
Cdevoff devclass; /* device type */
Cint devnum; /* device number */
```

`dissociate` removes the association between a trigger and a specified device. If `dissociate` is called while there are events pending in the event queue for the dissociated device, the pending events are discarded.

Errors

ENOTVSAC [4]	CGI not in proper state: CGI shall be in state VSAC.
EINDNOEX [80]	Input device does not exist.
EINDINIT [81]	Input device not initialized.
EINNTASD [85]	association does not exist.
EINTRNEX [86]	Trigger does not exist.

Set Initial Value

```
Cerror set_initial_value(devclass, devnum, value)
Cdevoff devclass; /* device type */
Cint devnum; /* device number */
Cinrep *value; /* device value */
```

`set_initial_value` sets the current measure of a specified device. This function resets the position of the track, if the track is appropriate and activated. `set_initial_value` also resets the request register.

A pointer element of the `Cinrep` structure must be set to the address of an application program allocated area before the values can be set. For example, in Figure 5-2 the following statements were necessary before an initial value could be assigned to the LOCATOR device.

```
Cinrep ivalue;
point.x = 16384;
point.y = 16384;
ivalue.xypt = &point;
```

Errors

ENOTVSAC [4]	CGI not in proper state: CGI shall be in state VSAC.
EINDNOEX [80]	Input device does not exist.
EINDINIT [81]	Input device not initialized.
EBADDATA [95]	Contents of input data record are invalid.
ESTRSIZE [96]	Length of initial string is greater than the implementation defined maximum.

Set VALUATOR Range

```
Cerror set_valuator_range(devnum, vmin, vmax)
Cint devnum; /* device number */
Cfloat vmin, vmax; /* limits of VALUATOR */
```

`set_valuator_range` specifies the limits of the IC_VALUATOR. Device coordinates are mapped into the IC_VALUATOR range. IC_VALUATOR events

which are already on the event queue are not rescaled. These events must be dequeued with either the `selective_flush_of_event_queue` function or `flush_event_queue`.

Errors

ENOTVSAC [4]	CGI not in proper state: CGI shall be in state VSAC.
EINDNOEX [80]	Input device does not exist.
EINDINIT [81]	Input device not initialized.

Track On

```
Cerror track_on(devclass, devnum, tracktype,
    trackregion, value)
Cdevoff devclass; /* device type */
Cint devnum; /* device number */
Cint tracktype; /* track number */
Ccoorpair *trackregion; /* window for tracking */
Cinrep *value; /* device value */
```

Tracking functions determine how the measure of an input device is displayed on the view surface. Each class of devices has its own set of possible tracks (given in Table 5-3). Although SunCGI allows certain classes of devices to track simultaneously, all types of input devices are not allowed to track at once. Tracking is not provided in the NO_EVENTS state unless the track type is PRINTERS_FIST.

`track_on` initiates track (or echo) for a specific device. The *tracktype* argument specifies the type of track to be used. The *trackregion* argument is not used; the device tracks in all areas of the view surface. The argument *value* is used to initialize tracking. The track is initially displayed on the first view surface opened.

The *xypt* element of the `Cinrep` structure must be set to the address of an application allocated `Ccoor` and the `Ccoor`'s *x* and *y* fields are set to position the cursor. The reference point for IC_STROKE echos 2 through 5 is the first point in the STROKE array. The reference point for STRING_TRACK echo is the `append_text` concatenation point, and can be changed by calling `text` or `append_text`.

Errors

ENOTVSAC [4]	CGI not in proper state: CGI shall be in state VSAC.
EINECHON [88]	Track already on.
EINETNSU [91]	Track type not supported.
EBADDATA [95]	Contents of input data record are invalid.
ESTRSIZE [96]	Length of initial string is greater than the implementation defined maximum.

Table 5-3 *Available Track Types*

Device Class	*Number*	*Track Type*†	*Description*
IC_LOCATOR	≤0	NO_ECHO	Default cursor.
	1	PRINTERS_FIST	Designate the current position of the IC_LOCATOR device with a printer's fist cursor.
IC_STROKE	≤0	NO_ECHO	Default cursor.
	1	PRINTERS_FIST	Designate the current position of the IC_STROKE device with a printer's fist cursor.
	2	SOLID_LINE	Draw a line from the origin to the current position in the STROKE array.
	3	X_LINE	Draw a line from the x-axis to the current position in the STROKE array.
	4	Y_LINE	Draw a line from the y-axis to the current position in the STROKE array.
	5	RUBBER_BAND_BOX	Designate the current position of the IC_STROKE device with a rubber band line connecting the initial position and the current position in the STROKE array.
IC_VALUATOR	≤0	NO_ECHO	Default cursor.
	1	PRINTERS_FIST	Indicate the state of the IC_VALUATOR device with a printer's fist cursor.
	2	STRING_TRACK	Display a digital representation of the current IC_VALUATOR value.
IC_CHOICE	≤0	NO_ECHO	Default cursor.
	1	PRINTERS_FIST	Indicate the state of the IC_CHOICE device with a printer's fist cursor.
IC_STRING	≤0	NO_ECHO	Default cursor.
	1	PRINTERS_FIST	Indicate the state of the IC_STRING device with a printer's fist cursor.
	2	STRING_TRACK	Display the current STRING value.

† The values listed in the *Track Type* column in Table 5-3 are contained in the enumerated type `Cechotype` returned in the `Cstatelist` structure by `inquire_lid_state_list`. They are *not* used by `track_on` to define a track type.

Track Off

```
Cerror track_off(devclass, devnum, tracktype, action)
Cdevoff devclass; /* device type */
Cint devnum; /* device number */
Cint tracktype;
Cfreeze action;
```

`track_off` terminates tracking for a specified input device. The *tracktype* and the *action* arguments are always ignored.

Errors

`ENOTVSAC` [4]	CGI not in proper state: CGI shall be in state VSAC.
`EINDNOEX` [80]	Input device does not exist.

EINDINIT [81] Input device not initialized.

5.2. Synchronous Input

The synchronous input function `request_input` allows the application program to obtain the current measure an of input device. This function requires explicit identification of an input device (through the `associate` function).

Figure 5-2 contains an example program that illustrates how to use the synchronous input functions to get information from an input device. First, a IC_LOCATOR device is initialized and associated with a trigger (the left mouse button). The tracking method for the IC_LOCATOR is defined to be a printer's fist. Then measure of the IC_LOCATOR is requested with a timeout period of ten seconds. If the trigger is activated during this period, `request_input` returns a valid measure in *ivalue*. Finally, the IC_LOCATOR is dissociated from the mouse button and released. The program exits.

```
#include <cgidefs.h>
#define TEN_SECONDS (10 * 1000 * 1000)

main()
{
    Cawresult stat;
    Ccoor point;
    Cinrep ivalue;
    Cint name;
    Cint trigger;
    Cvwsurf device;

    NORMAL_VWSURF(device, PIXWINDD);
    point.x = 16384;
    point.y = 16384;
    ivalue.xypt = &point;

    open_cgi();
    open_vws(&name, &device);

    initialize_lid(IC_LOCATOR, 1, &ivalue);
    associate(2, IC_LOCATOR, 1);
    track_on(IC_LOCATOR, 1, 1, (Ccoorpair *)0, &ivalue);
    request_input(IC_LOCATOR, 1, TEN_SECONDS,
        &stat, &ivalue, &trigger);
    if (stat == VALID_DATA)
        printf("trigger activated at %d %d \n",
            ivalue.xypt->x, ivalue.xypt->y);
    else
        printf("trigger not activated \n");
    dissociate(2, IC_LOCATOR, 1);
    release_input_device(IC_LOCATOR, 1);

    close_vws(name);
    close_cgi();
}
```

Figure 5-2 *Example Program with LOCATOR Input Device*

Request Input

```
Cerror request_input(devclass, devnum, timeout,
    valid, sample, trigger)
Cdevoff devclass; /* device type */
Cint devnum; /* device number */
Cint timeout; /* amount of time to wait for input */
Cawresult *valid; /* device status */
Cinrep *sample; /* device value */
Cint *trigger; /* trigger number */
```

`request_input` waits *timeout* microseconds for activation of a trigger associated with a specific device. If *timeout* is negative, the request will wait forever.

`request_input` puts the input device in the RESPOND_EVENT state. If a trigger is activated within this period, the activating trigger and the device measure are returned in the *trigger* and *sample* arguments respectively. If the trigger is not activated within this period, the current device measure is returned in the *sample* argument and *trigger* is set to zero. Before returning, the input device is reset to NO_EVENTS state.

`request_input` returns a device status in the argument *valid*. This argument uses the enumerated type `Cawresult` (AWait Result) which contains values describing the state of an input device.

```
typedef enum {
    VALID_DATA,
    TIMED_OUT,
    DISABLED,
    WRONG_STATE,
    NOT_SUPPORTED
} Cawresult;
```

VALID_DATA indicates a trigger is activated within the specified timeout period. TIMED_OUT indicates that a trigger was not activated with a specified period. WRONG_STATE indicates SunCGI is not in state VSAC. NOT_SUPPORTED indicates the requested device is not a legal device.

If the appropriate field of the *sample* argument is a pointer, it must be set to an application program allocated area.

Errors

`ENOTVSAC` [4]	CGI not in proper state: CGI shall be in state VSAC.
`EINDNOEX` [80]	Input device does not exist.
`EINDINIT` [81]	Input device not initialized.
`EINEVNEN` [94]	Events not enabled.

5.3. Asynchronous Input

This section explains the asynchronous method of input device management where the application process and the input device process operate simultaneously. The designated input device is sampled with `initiate_request` and the measure of the input device is read with `get_last_requested_input`. Alternatively, the current measure of a device may be read with `sample_input`.

The example program in Figure E-2 demonstrates how to use the asynchronous input functions.

Initiate Request

```
Cerror initiate_request(devclass, devnum)
Cdevoff devclass; /* device type */
Cint devnum; /* device number */
```

`initiate_request` sets up a device so that the measure resulting from the next trigger activation will be placed in the request register. `initiate_request` puts the device in the REQUEST_EVENT state. It then returns to the calling function without waiting for a trigger activation. The value caused by the trigger activation can be obtained by the

`get_last_requested_input` function.

Errors

`ENOTVSAC` [4]	CGI not in proper state: CGI shall be in state VSAC.
`EINDNOEX` [80]	Input device does not exist.
`EINDINIT` [81]	Input device not initialized.
`EINNTASD` [85]	No triggers associated with device.

5.4. Event Queue Input

The event queue is a single FIFO buffer that holds events from input devices. Since the event queue has a fixed length, it must be processed in a timely fashion or it will overflow. Events can be removed from the event queue in three ways: the event at the head of the event queue can be processed with `await_event`; the entire event queue can be emptied with `flush_event_queue`; and the events from a particular device can be removed from the event queue with `selective_flush_of_event_queue`.

Figure 5-3 contains an example program that illustrates how to use the event queue input functions to get information from an input device. First, a IC_STRING device is initialized and associated with a trigger (the keyboard). The tracking method for the IC_STRING is defined to be a string that echos the keyboard input on the bottom of the viewport. The IC_STRING is put into the QUEUE_EVENT state with `enable_events`. After the trigger fires, the measure of the IC_STRING device is determined with `await_event`. Finally, the LOCATOR is dissociated from the mouse button and released. The program then exits.

```
#include <cgidefs.h>

main()
{
    Cawresult valid;
    Ccoor point;
    Cdevoff devclass = IC_STRING;
    Ceqflow overflow;
    Cinrep ivalue;
    Cint devnum = 1;
    Cint name;
    Cint replost;
    Cint time_stamp;
    Cint timeout = (10 * 1000 * 1000); /* ten seconds */
    Cint tracktype = 2;
    Cint trigger = 1;
    Cmesstype message_link;
    Cqtype qstat;
    Cvwsurf device;

    NORMAL_VWSURF(device, PIXWINDD);
    point.x = 16384;
    point.y = 16384;
    ivalue.xypt = &point;
    ivalue.string = "This is a string";

    open_cgi();
    open_vws(&name, &device);

    initialize_lid(devclass, devnum, &ivalue);
    associate(trigger, devclass, devnum);
    track_on(devclass, devnum, tracktype,
        (Ccoorpair *)0, &ivalue);
    enable_events(devclass, devnum);
    await_event(timeout, &valid, &devclass, &devnum,
        &ivalue, &message_link, &replost, &time_stamp,
        &qstat, &overflow);
    printf("%s\n", ivalue.string);
    disable_events(IC_STRING, devnum);
    dissociate(trigger, IC_STRING, devnum);
    release_input_device(IC_STRING, devnum);

    close_vws(name);
    close_cgi();
}
```

Figure 5-3 *Example Program with STRING Input Device*

Enable Events

```
Cerror enable_events(devclass, devnum)
Cdevoff devclass; /* device type */
Cint devnum; /* device number */
```

`enable_events` allows a device in NO_EVENTS state to put events on the event queue. `enable_events` puts the input device in the QUEUE_EVENT state. An error is generated if the device specified by *devclass* or *devnum* does not exist or is not initialized.

Errors

ENOTVSAC [4]	CGI not in proper state: CGI shall be in state VSAC.
EINDNOEX [80]	Input device does not exist.
EINDINIT [81]	Input device not initialized.
EIAEVNEN [93]	Events already enabled.

Await Event

```
Cerror await_event(timeout, valid, devclass, devnum,
    measure, message_link, replost, time_stamp,
    qstat, overflow)
Cint timeout; /* input timeout period */
Cawresult *valid; /* status */
Cdevoff *devclass; /* device type */
Cint *devnum; /* device number */
Cinrep *measure; /* device value */
Cmesstype *message_link; /* type of message */
Cint *replost; /* reports lost */
Cint *time_stamp; /* time_stamp */
Cqtype *qstat; /* queue status */
Ceqflow *overflow; /* event queue status */
```

`await_event` processes the event at the head of the event queue. *valid* is set to WRONG_STATE if SunCGI is not in state VSAC. If the event queue is EMPTY, then `await_event` waits *timeout* microseconds for a trigger to be activated. If *timeout* is less than 0, SunCGI waits until a trigger is activated. *valid* is set to VALID_DATA if a trigger is activated within the specified timeout period and TIMED_OUT otherwise.

If either the event queue is not empty or a trigger is activated, the class, number and value of the device generating the event are reported in the returned arguments *devclass*, *devnum and measure*. *If the appropriate field of the measure argument is a pointer, it must be set to an application program allocated area.*

If two events on the event queue have the same trigger but different values, the argument *message_link* is assigned to SIMULTANEOUS_EVENT_FOLLOWS; otherwise the argument *message_link* is set to SINGLE_EVENT. The enumerated type `Cmesstype` contains the following values:

```
typedef enum {
    SIMULTANEOUS_EVENT_FOLLOWS,
    SINGLE_EVENT
} Cmesstype;
```

The *replost* and *time_stamp* arguments should be ignored and are always zero. The returned argument *qstat* reports the queue status after an event is removed

from the head of the event queue.

```
typedef enum {
    NOT_VALID,
    EMPTY,
    NON_EMPTY,
    ALMOST_FULL,
    FULL
} Cqtype;
```

qstat is set to EMPTY if the event queue has no pending events. *qstat* is set to NON_EMPTY if the event queue has events pending, but is not FULL or ALMOST_FULL. *qstat* is set to ALMOST_FULL if there is room for only one more event on the event queue. *qstat* is set to FULL if there is no room for more events on the event queue.

The argument *overflow* indicates whether the event queue has overflowed or not. The enumerated type `Ceqflow` contains the following values:

```
typedef enum {
    NO_OFLO,
    OFLO
} Ceqflow;
```

Errors

ENOTVSAC [4]	CGI not in proper state: CGI shall be in state VSAC.
EINQOVFL [97]	Input queue has overflowed.

Flush Event Queue

```
Cerror flush_event_queue()
```

`flush_event_queue` discards all events in the event queue. The purpose of `flush_event_queue` is to return the event queue to a stable state (NO_OFLO). `flush_event_queue` does not affect the state of input devices. This function should be used carefully to avoid throwing away mouse-ahead or type-ahead inputs.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in either in state VDOP, VSOP, or VSAC.

Selective Flush of Event Queue

```
Cerror selective_flush_of_event_queue(devclass, devnum)
Cdevoff devclass; /* device type */
Cint devnum; /* device number */
```

`selective_flush_of_event_queue` discards all events in the event queue which were generated by a specified device. `selective_flush_of_event_queue` does not affect the state of the specified input device. *devclass* and *devnum* must refer to an existing and initialized device or an error is produced. However, no error is returned if no events from the specified device are pending.

Errors

ENOTOPOP [5]	CGI not in proper state CGI shall be in either in state VDOP, VSOP, or VSAC.

EINDNOEX [80]	Input device does not exist.
EINDINIT [81]	Input device not initialized.

5.5. Miscellaneous Input Functions

The functions described in this section can be used with several of the input device management techniques described in the previous sections. For example, `sample_input` can be used when a device is in either RESPOND_EVENT or QUEUE_EVENT state. Likewise, `disable_events` can be used in either of these states.

Sample Input

```
Cerror sample_input(devclass, devnum, valid, sample)
Cdevoff devclass; /* device type */
Cint devnum; /* device number */
Clogical *valid; /* device status */
Cinrep *sample; /* device value */
```

`sample_input` reports the current measure of the specified input device in the returned argument *sample*. The returned argument *valid* reports whether the device is initialized and prepared to receive an input. The current measure of the device may be set by a queued event, a requested event, or a device initialization depending on the state of the input device and the most recent trigger activation(s). See the introduction of this chapter for an explanation of the relationship between the *measure* of an input device and the *state* of an input device. If the appropriate field of the *sample* argument is a pointer, it must be set to an application program allocated area.

Errors

ENOTVSAC [4]	CGI not in proper state: CGI shall be in state VSAC.
EINDNOEX [80]	Input device does not exist.
EINDINIT [81]	Input device not initialized.

Get Last Requested Input

```
Cerror get_last_requested_input(devclass, devnum,
    valid, sample)
Cdevoff devclass; /* device type */
Cint devnum; /* device number */
Clogical *valid; /* device status */
Cinrep *sample; /* device value */
```

`get_last_requested_input` returns the contents of the request register. `get_last_requested_input` is usually used with `initiate_request`, but `request_input` also changes the contents of the request register. The returned argument *valid* indicates whether the device exists and is initialized. The returned argument *sample* reports the event in the request register. If no event is in the request register, the initial device value is reported. If the appropriate field of the *sample* argument is a pointer, it must be set to an application program allocated area.

Errors

ENOTVSAC [4]	CGI not in proper state: CGI shall be in state VSAC.
EINDNOEX [80]	Input device does not exist.

EINDINIT [81]	Input device not initialized.

Disable Events

```
Cerror disable_events(devclass, devnum)
Cdevoff devclass; /* device type */
Cint devnum; /* device number */
```

`disable_events` puts the input device in the NO_EVENTS state. If the device is in RESPOND_EVENT state, the specified device is returned to NO_EVENTS state; the measure of the device is not changed by `disable_events`. If the device is in QUEUE_EVENT state, `disable_events` stops the specified device from putting events on the event queue. However, existing entries on the event queue are not removed and existing associations remain. *devclass* and *devnum* must refer to an existing and initialized device or an error is produced.

Errors

ENOTVSAC [4]	CGI not in proper state: CGI shall be in state VSAC.
EINDNOEX [80]	Input device does not exist.
EINDINIT [81]	Input device not initialized.
EINEVNEN [94]	Events not enabled.

5.6. Status Inquiries

The current state of the input devices, triggers, and the event queue can be obtained by using the functions discussed in this section.

Inquire LID State List

```
Cerror inquire_lid_state_list(devclass, devnum,
    valid, list)
Cdevoff devclass; /* device type */
Cint devnum; /* device number */
Clogical *valid; /* device supported at all */
Cstatelist *list; /* table of descriptors */
```

`inquire_lid_state_list` reports the status of a specific input device specified by *devclass* and *devnum*. The argument *valid* reports whether the device is supported at all. The *list* argument reports the track, associations, state and measure of the device in the appropriate elements of *list*. When checking the elements of *list*, first check the *state* element — if *state* is RELEASED, the other elements of *list* are undefined.

```
typedef struct {
    Clidstate state;
    Cpromstate prompt;
    Cackstate acknowledgement;
    Cinrep *current;
    Cint n;
    Cint *triggers;
    Cechotype echotyp;
    Cechostate echosta;
    Cint echodat;
} Cstatelist;
```

Errors

ENOTVSAC [4]	CGI not in proper state: CGI shall be in state VSAC.
EINDNOEX [80]	Input device does not exist.

Inquire LID State

```
Cerror inquire_lid_state(devclass, devnum, valid, state)
Cdevoff devclass; /* device type */
Cint devnum; /* device number */
Clogical *valid; /* device supported at all */
Clidstate *state; /* table of descriptors */
```

`inquire_lid_state` reports the status of a specific input device specified by *devclass* and *devnum*. The argument *valid* reports whether the device is supported at all. The *state* argument (of type `Clidstate`) reports the current state of the specified input device.

```
typedef enum {
    RELEASE,
    NO_EVENTS,
    REQUEST_EVENT,
    RESPOND_EVENT,
    QUEUE_EVENT
} Clidstate;
```

Errors

ENOTVSAC [4]	CGI not in proper state: CGI shall be in state VSAC.
EINDNOEX [80]	Input device does not exist.

Inquire Trigger State

```
Cerror inquire_trigger_state(trigger, valid, list)
Cint trigger; /* trigger number */
Clogical *valid; /* trigger state */
Ctrigstate *list; /* trigger description table */
```

`inquire_trigger_state` describes the binding between a trigger and an input device. If the *state* element of the returned argument *list* is INACTIVE, no associations have been made with the trigger. An error is generated if the trigger does not exist.

```
typedef struct {
    Cactstate state; /* state */
    Cassoclid *assoc; /* list of associations */
} Ctrigstate;
```

Errors

ENOTVSAC [4]	CGI not in proper state: CGI shall be in state VSAC.
EINTRNEX [86]	Trigger does not exist.

Inquire Event Queue State

```
Cerror inquire_event_queue_state(qstat, qflow)
Cqtype * qstat; /* queue state */
Ceqflow * qflow; /* overflow indicator */
```

`inquire_event_queue_state` reports the status of the event queue. *qstat* indicates whether any events are pending. The argument *qflow* reports if the event queue is overflowing.

```
typedef enum {
    NOT_VALID,
    EMPTY,
    NON_EMPTY,
    ALMOST_FULL,
    FULL
} Cqtype;

typedef enum {
    NO_OFLO,
    OFLO
} Ceqflow;
```

Errors

ENOTVSAC [4] CGI not in proper state: CGI shall be in state VSAC.

A

Differences between SunCore and SunCGI

A

Differences between SunCore and SunCGI

This appendix provides an introduction to SunCGI for programmers who have programming experience with SunCore or graphics packages based on the ACM Core Graphics Specification. The three major differences between SunCore and SunCGI are in the areas of output primitives, segmentation, and input. While SunCore is generally a 'higher-level' package, SunCGI has capabilities which are not available in SunCore.

A.1. Output Primitives

The major differences in drawing objects to the screen between SunCore and SunCGI are that

1. SunCGI does not support 3D primitives, and
2. SunCGI does not have floating-point world coordinates or image transforms, and,
3. SunCGI does not support the concept of current position, and
4. SunCGI does not support textured color lookup table for monochrome devices.

However, SunCGI provides a wider variety of geometrical and raster primitives, and more control over the drawing of text. These differences are summarized in Table A-1.

Table A-1 *Difference in Output Primitives*

Feature	*SunCore*	*SunCGI*
3D Output Primitives	Yes	No
Current Position	Yes	No
Textured Color Lookup Tables	Yes	No
Polygons with Invisible Edges	No	Yes
Circles and Ellipses	No	Yes
Cell Arrays	No	Yes
Character Clipping	No	Yes

Output Aspects of SunCore not Supported by SunCGI

SunCGI does not support 3D output primitives, current position, or textured color lookup tables for monochrome devices. Since 3D output primitives are not supported, no shading or lighting functions are provided either. Furthermore, no rotation or translation functions are provided. Therefore, if you want to rotate a geometrical output primitive, these operations must be done by your application program.

Since SunCGI does not maintain the current position of the output 'cursor', relative drawing functions such as `polygon_rel_3` are not supported. However, the application programmer can implement this function by specifying all coordinates as a base register plus a constant. The base register can be used by the application program to maintain the value of the current position.

For monochrome devices, SunCore interprets the entries in the color lookup table with indices greater than one as patterns. SunCGI interprets all color lookup table entries greater than zero as black. Patterns in SunCGI are explicitly specified in the pattern table and invoked by using the PATTERN or HATCH interior styles. In addition, while patterns in SunCore are all 4×4 matrices, patterns in SunCGI have variable dimensions.

Output Features of SunCGI not Available in SunCore

SunCGI offers geometrical and raster primitives not available in SunCore, as well as increased control over the drawing of text. SunCGI provides circles and ellipses. SunCGI also supports the cell array which is a raster array whose element size is a function of the screen size. SunCGI clips characters in parts if the *text precision* is set to STROKE.

A.2. Segmentation

SunCGI does not support segmentation. This effect influences the effect of attribute calls. In SunCore, some attributes (for example, highlighting) apply to entire segments. Since no concept of segmentation exists in SunCGI, these attributes are not offered. Furthermore, SunCGI does not allow the saving or restoring of segments to the screen, so screen repainting functions must be completely defined by the application program, unless the view surface is initialized as a retained view surface and is not resized.

A.3. Differences in Input Functions between SunCore and SunCGI

SunCore provides device-specific functions for setting input device parameters and reading input from them. SunCGI provides no device dependent calls. SunCGI has three methods for obtaining the measure of input devices

1. by first activation (REQUEST EVENT),
2. by most recent activation (RESPOND EVENT), or
3. by mediating input requests through the *event queue* (QUEUE EVENT).

Furthermore, SunCGI allows the explicit binding of triggers (physical input devices) to logical input devices.

B

Unsupported Aspects of CGI

B

Unsupported Aspects of CGI

SunCGI does not support certain optional aspects of the proposed draft ANSI CGI standard. Most notably SunCGI does not support the full constellation of negotiation functions or tracking. SunCGI does not allow the resetting of *coordinate type*, *coordinate precision* or *color specification mode* because to do so would greatly reduce the speed of application programs written in SunCGI. Furthermore, SunCGI does not support echoing functions for input, but provides the tracking functions instead.

Table B-1 *Unsupported Control Functions*

Function
`vdc_type`
`vdc_precision_for_integer_points`
`vdc_precision_for_real_points`
`integer_precision`
`real_precision`
`index_precision`
`color_selection_mode`
`color_precision`
`color_index_precision`
`viewport_specification_mode`
`make_picture_current`

Table B-2 *Unsupported Input Functions*

Function
`set_prompt_state`
`set_acknowledgement_state`
`echo_on`
`echo_off`
`echo_update`

The following SunCGI functions are nonstandard (that is, are not in the standards document) and are included to make CGI easier to use. In addition, SunCGI has non-standard view surface arguments for certain control functions.

Table B-3 *Non Standard Control Functions*

Function
`open_cgi`
`open_vws`
`activate_vws`
`deactivate_vws`
`close_vws`
`close_cgi`

Table B-4 *Non Standard Attribute Functions*

Function
`define_bundle_index`
`line_endstyle`
`set_global_drawing_mode`
`pattern_with_fill_color`
`fixed_font`

The `Cinrep` structure contains a presently unsupported *pick* field, for compatibility with future segment manipulation capabilities.

C

Type and Structure Definitions

C

Type and Structure Definitions

This appendix provides a list of the structures and enumerated types used by SunCGI functions. In addition, a list of useful constants defined in <cgiconstants.h> is given.

```
    /*devices*/
#define BW1DD 1
#define BW2DD 2
#define CG1DD 3
#define PIXWINDD 4
#define CGPIXWINDD 5
#define GP1DD 6
#define CG2DD 7

#define VWSURF_NEWFLG 1

    /* limits */
#define MAXVWS 5
#define MAXTRIG 6
#define MAXASSOC 5
#define MAXEVENTS 1024
#define MAXAESSIZE 10 /* maximum number of AES table entries */
#define MAXNUMPATS 50 /* maximum number of pattern table entries */
#define MAXPATSIZE 256 /* maximum pattern size */
#define MAXPTS 1024 /* maximum number of pts per polygon */
#define MAXCHAR 256 /* maximum number of chars in a string */
#define OUTFUNS 67 /* number of output functions */
#define INFUNS 22 /* number of input functions */
#define SMALL_CHAR 6 /* minimum character size */
#define DEVNAMESIZE 20
```

The type and structure definitions that follow can be found in the header file <cgidefs.h>.

```
typedef enum {
    ACK_ON,
    ACK_OFF
} Cackstate;

typedef enum {
    ACTIVE,
    INACTIVE
```

```
} Cactstate;

typedef enum {
    CLEAR,
    NO_OP,
    RETAIN
} Cacttype;

typedef enum {
    INDIVIDUAL,
    BUNDLED
} Casptype;

typedef struct {
    Cint n;
    Cdevoff *class;
    Cint *assoc;
} Cassoclid;

typedef enum {
    VALID_DATA,
    TIMED_OUT,
    DISABLED,
    WRONG_STATE,
    NOT_SUPPORTED
} Cawresult;

typedef enum {
    BITNOT,
    BITTRUE
} Cbitmaptype;

typedef enum {
    TRANSPARENT,
    OPAQUE
} Cbmode;

typedef struct {
    Clintype line_type;
    Cfloat line_width;
    Cint line_color;
    Cmartype marker_type;
    Cfloat marker_size;
    Cint marker_color;
    Cintertype interior_style;
    Cint hatch_index;
    Cint pattern_index;
    Cint fill_color;
    Clintype perimeter_type;
    Cfloat perimeter_width;
    Cint perimeter_color;
    Cint text_font;
    Cprectype text_precision;
```

```
    Cfloat character_expansion;
    Cfloat character_spacing;
    Cint text_color;
} Cbunatt;

typedef struct {
    unsigned char *ra;
    unsigned char *ga;
    unsigned char *ba;
    Cint n;
} Ccentry;

typedef enum {
    OPEN,
    CLOSE
} Ccflag;

typedef struct {
    Cint numloc;
    Cint numval;
    Cint numstrk;
    Cint numchoice;
    Cint numstr;
    Cint numtrig;
    Csuptype event_queue;
    Csuptype asynch;
    Csuptype coord_map;
    Csuptype echo;
    Csuptype tracking;
    Csuptype prompt;
    Csuptype acknowledgement;
    Csuptype trigger_manipulation;
} Ccgidesctab;

typedef enum {
    YES,
    NO
} Cchangetype;

typedef char Cchar;

typedef enum {
    NOCLIP,
    CLIP,
    CLIP_RECTANGLE
} Cclip;

typedef enum {
    CHORD,
    PIE
} Cclosetype;

typedef enum {
```

```
    REPLACE,
    AND,
    OR,
    NOT,
    XOR
} Ccombtype;

typedef struct {
    Cint x;
    Cint y;
} Ccoor;

typedef struct {
    Ccoor *ptlist;
    Cint n;
} Ccoorlist;

typedef struct {
    Ccoor *upper;
    Ccoor *lower;
} Ccoorpair;

typedef enum {
    IC_LOCATOR,
    IC_STROKE,
    IC_VALUATOR,
    IC_CHOICE,
    IC_STRING,
    IC_PICK
} Cdevoff;

typedef enum {
    E_TRACK,
    E_ECHO,
    E_TRACK_OR_ECHO,
    E_TRACK_AND_ECHO
} Cechoav;

typedef struct {
    Cinrep *echos;
    Cint n;
} Cechodatalst;

typedef enum {
    ECHO_OFF,
    ECHO_ON,
    TRACK_ON
} Cechostate;

typedef struct {
    Cechostate *echos;
    Cint n;
} Cechostatelst;
```

```
typedef enum {
    NO_ECHO,
    PRINTERS_FIST,
    HIGHLIGHT,
    RUBBER_BAND_BOX,
    DOTTED_LINE,
    SOLID_LINE,
    STRING_ECHO,
    XLINE,
    YLINE
} Cechotype;

typedef struct {
    Cint n;
    Cechoav *elements;
    Cechotype *echos;
} Cechotypelst;

typedef enum {
    NATURAL,
    POINT,
    BEST_FIT
} Cendstyle;

typedef enum {
    NO_OFLO,
    OFLO
} Ceqflow;

typedef Cint Cerror;

typedef enum {
    INTERRUPT,
    NO_ACTION,
    POLL
} Cerrtype;

typedef enum {
    CLIP_RECT,
    VIEWPORT,
    VIEWSURFACE
} Cexttype;

typedef struct {
    Cintertype style;
    Cflag visible;
    Cint color;
    Cint hatch_index;
    Cint pattern_index;
    Cint index;
    Clintype pstyle;
    Cfloat pwidth;
    Cint pcolor;
```

```
} Cfillatt;

typedef enum {
    OFF,
    ON
} Cflag;

typedef struct {
    Cint n;
    Cint *num;
    Casptype *value;
} Cflaglist;

typedef float Cfloat;

typedef enum {
    FREEZE,
    REMOVE
} Cfreeze;

typedef enum {
    LFT,
    CNTER,
    RGHT,
    NRMAL,
    CNT
} Chaligntype;

typedef enum {
    NO_INPUT,
    ALWAYS_ON,
    SETTABLE,
    DEPENDS_ON_LID
} Cinputability;

typedef struct {
    Ccoor *xypt; /* LOCATOR */
    Ccoorlist *points; /* STROKE devices */
    Cfloat val; /* VALUATOR device */
    Cint choice; /* CHOICE devices */
    Cchar *string; /* STRING device */
    Cpick *pick; /* PICK devices */
} Cinrep;

typedef int Cint;

typedef enum {
    HOLLOW,
    SOLIDI,
    PATTERN,
    HATCH
} Cintertype;
```

```
typedef struct {
    Clogical sample;
    Cchangetype change;
    Cint numassoc;
    Cint *trigassoc;
    Clogical prompt;
    Clogical acknowledgement;
    Cechotypelst *echo;
    Cchar *classdep;
    Cstatelist state;
} Cliddescript;

typedef enum {
    RELEASE,
    NO_EVENTS,
    REQUEST_EVENT,
    RESPOND_EVENT,
    QUEUE_EVENT
} Clidstate;

typedef struct {
    Clintype style;
    Cfloat width;
    Cint color;
    Cint index;
} Clinatt;

typedef enum {
    SOLID,
    DOTTED,
    DASHED,
    DASHED_DOTTED,
    DASH_DOT_DOTTED,
    LONG_DASHED
} Clintype;

typedef enum {
    L_FALSE,
    L_TRUE
} Clogical;

typedef struct {
    Cmartype type;
    Cfloat size;
    Cint color;
    Cint index;
} Cmarkatt;

typedef enum {
    DOT,
    PLUS,
    ASTERISK,
    CIRCLE,
```

```
    X
} Cmartype;

typedef enum {
    SIMULTANEOUS_EVENT_FOLLOWS,
    SINGLE_EVENT
} Cmesstype;

typedef enum {
    RIGHT,
    LEFT,
    UP,
    DOWN
} Cpathtype;

typedef struct {
    Cint cur_index;
    Cint row;
    Cint column;
    Cint *colorlist;
    Ccoor *point;
    Cint dx;
    Cint dy;
} Cpatternatt;

typedef struct {
    int segid; /* segment */
    int pickid; /* pick id */
} Cpick;

typedef struct pixrect Cpixrect;

typedef enum {
    STRING,
    CHARACTER,
    STROKE
} Cprectype;

typedef enum {
    PROMPT_OFF,
    PROMPT_ON
} Cpromstate;

typedef enum {
    NOT_VALID,
    EMPTY,
    NON_EMPTY,
    ALMOST_FULL,
    FULL
} Cqtype;

typedef enum {
    ABSOLUTE,
```

```
    SCALED
} Cspecmode;

typedef struct {
    Clidstate state;
    Cpromstate prompt;
    Cackstate acknowledgement;
    Cinrep *current;
    Cint n;
    Cint *triggers;
    Cechotype echotyp;
    Cechostate echosta;
    Cint echodat;
} Cstatelist;

typedef enum {
    NONE,
    REQUIRED_FUNCTIONS_ONLY,
    SOME_NON_REQUIRED_FUNCTIONS,
    ALL_NON_REQUIRED_FUNCTIONS
} Csuptype;

typedef struct {
    Cint fontset;
    Cint index;
    Cint current_font;
    Cprectype precision;
    Cfloat exp_factor;
    Cfloat space;
    Cint color;
    Cint height;
    Cfloat basex;
    Cfloat basey;
    Cfloat upx;
    Cfloat upy;
    Cpathtype path;
    Chaligntype halign;
    Cvaligntype valign;
    Cfloat hcalind;
    Cfloat vcalind;
} Ctextatt;

typedef enum {
    NOT_FINAL,
    FINAL
} Ctextfinal;

typedef struct {
    Cchangetype change;
    Cassoclid *numassoc;
    Cint maxassoc;
    Cpromstate prompt;
    Cackstate acknowledgement;
```

```
    Cchar *name;
    Cchar *description;
} Ctrigdis;

typedef struct {
    Cactstate state;
    Cassoclid *assoc;
} Ctrigstate;

typedef enum {
    TOP,
    CAP,
    HALF,
    BASE,
    BOTTOM,
    NORMAL,
    CONT
} Cvaligntype;

typedef enum {
    INTEGER,
    REAL,
    BOTH
} Cvdctype;

typedef struct {
    Cchar screenname[DEVNAMESIZE]; /* physical screen */
    Cchar windowname[DEVNAMESIZE]; /* window */
    Cint windowfd; /* window file */
    Cint retained; /* retained flag */
    Cint dd; /* device */
    Cint cmapsize; /* color map size */
    Cchar cmapname[DEVNAMESIZE]; /* color map name */
    Cint flags; /* new flag */
    Cchar **ptr; /* CDI tool descriptor */
} Cvwsurf;
```

D

Error Messages

D

Error Messages

This appendix lists the error messages in numerical order. Furthermore, the probable cause of each error is given in the sentences following the error. In addition to explaining the error message, an initial suggestion for corrective action is given. In the title for each group of errors, the range of error numbers is given in parentheses after the title. If your application program is not behaving as you want it to, but does not generate error messages, then the table at the end of this appendix which lists commonly encountered problems and frequent causes may be helpful.

D.1. Successful Return (0)

`NO_ERROR [0]` — *No error*.

D.2. State Errors (1-5)

`ENOTCGCL [1]` — *CGI not in proper state: CGI should be in state CGCL*. A call to `open_cgi` was attempted when cgi was already open. Elimination of the error can be accomplished by removing the offending call to `open_cgi`.

`ENOTCGOP [2]` — *CGI not in proper state: CGI should be in state CGOP*. Every function except `open_cgi` requires that CGI be open. If this error is received, make sure that your application program has called `open_cgi`, or that it has not recently called `close_cgi`

`ENOTVSOP [3]` — *CGI not in proper state: CGI should be in state VSOP*. The function which generated the error requires that at least one view surface be open. Corrective action would include either removing the most recent call to `close_vws` or by including a call to `open_vws`.

`ENOTVSAC [4]` — *CGI not in proper state: CGI should be in state VSAC*. The function which generated the error requires that at least one view surface be active. Corrective action would include either removing the most recent call to `deactivate_vws` or by including a call to `activate_vws`.

`ENOTOPOP [5]` — *CGI not in proper state CGI should be in state CGOP, VSOP, or VSAC*. The function which generated the error requires that SunCGI is at least initialized. If this error is received, make sure that your application program has

		called `open_cgi`, or that it has not recently called `close_cgi`.
D.3. Control Errors (10-16)	EVSIDINV [10]	*Specified view surface name is invalid*. The view surface name specified by the *name* argument has never been opened or if it has been opened, it has since been closed. Corrective action involves opening the view surface or changing the value of the *name* argument.
	ENOWSTYP [11]	*Specified view surface type does not exist*. The application program has specified a type of view surface which is not supported by SunCGI. Corrective action involves changing the type of view surface.
	EMAXVSOP [12]	*Maximum number of view surfaces already open*. An attempt was made to open a view surface when the maximum number of view surfaces is already open. Corrective action involves removing one call to `open_vws`.
	EVSNOTOP [13]	*Specified view surface not open*. An attempt was made to close a view surface which is already closed. Corrective action involves removing one call to `close_vws`.
	EVSISACT [14]	*Specified view surface is active*. An attempt was made to activate a view surface which is already activated. Corrective action involves removing one call to `activate_vws`.
	EVSNTACT [15]	*Specified view surface is not active*. An attempt was made to deactivate a view surface which has already been deactivated. Corrective action involves removing one call to `deactivate_vws`.
	EINQALTL [16]	*Inquiry arguments are longer than list*. A call to inquiry negotiation function with indices greater than the number of supported functions was made. The returned list is always empty. Corrective action may be facilitated by obtaining the size of the list by using the `inquire_device_class` function.
D.4. Coordinate Definition (20-24)	EBADRCTD [20]	*Rectangle definition is invalid*. The application program has made a call to `vdc_extent` or `device_viewport` with the coordinates of both corners equal in the x or y dimensions or both. Corrective action involves changing one of the arguments to the function which generated the error so that the values of the two arguments are different in both the x and y dimensions.
	EBDVIEWP [21]	*Viewport is not within Device Coordinates*. A call to `device_viewport` has been made which specifies a viewport which is larger than the view surface. Corrective action involves making the arguments to

`device_viewport` less than the view surface size. The size of the view surface can be obtained by calling the `inquire_physical_coordinate_system` function.

`ECLIPTOL` [22] *Clipping rectangle is too large*. The clipping rectangle would exceed the boundaries of VDC space. Corrective action involves resetting the clipping rectangle to be within limits of VDC space.

`ECLIPTOS` [23] *Clipping rectangle is too small*. The clipping rectangle would define an area of screen space smaller than one pixel. The clipping rectangle remains unchanged. Since the occurrence of this error is partially a function of the size of the view surface, changing the size of the view surface may be a viable alternative to changing the size of the clipping rectangle.

`EVDCSDIL` [24] *VDC space definition is illegal*. One or more of the arguments to the `vdc_extent` function exceeds the acceptable limits (-32767 to 32767) or coordinates of the lower-left hand corner are greater than the coordinates of the upper-right hand corner. Corrective action involves changing the arguments to `vdc_extent`.

D.5. Output Attributes (30-51)

`EBTBUNDL` [30] *ASF is BUNDLED*. Error 16 is generated when attempting to call an individual attribute function when the attributes are specified by entries in the *attribute environment table*. Calls to these functions have no effect on the current attributes. Corrective action includes resetting the *attribute environment selector* to BUNDLED by using the `set_attribute_environment_selector` function.

`EBBDTBDI` [31] *Bundle table index out of range*. The entry in the *bundle table* exceeds the size of the table. The only corrective action is to change the value of the *index* argument.

`EBTUNDEF` [32] *Bundle table index is undefined*. The entry in the *attribute environment table* specified by the most recent call to `set_attribute_environment_table_index` has not been defined by SunCGI or the application program.

`EBADLINX` [33] *Polyline index is invalid*. The polyline bundle is not defined. Corrective action involves changing the *index* argument to `polyline_bundle_index`, or by defining the polyline bundle index.

`EBDWIDTH` [34] *Width must be nonnegative*. The width of a perimeter or line must be greater than or equal to zero. The current value of the *perimeter width* or *line width* remains

unchanged. Changing the value of the width argument to a non-negative value will correct this error.

ECINDXLZ [35] *Color index is less than zero*. The value of the *index* argument to one of the attribute functions or the color entry in one of the bundles is negative. Corrective action involves changing the value of the color.

EBADCOLX [36] *Color index is invalid*. The color index argument to one of the attribute functions or the color entry in one of the bundles is not defined in the colormap. Indices in the *color lookup table* must be between 0 and 255 for the Sun 8-bit per pixel frame buffer. Any color specification outside of this range is ignored. Corrective action involves changing the value of the color.

EBADMRKX [37] *Polymarker index is invalid*. The polymarker bundle is not defined. Corrective action involves changing the *index* argument to `polymarker_bundle_index`, or by defining the polymarker bundle index.

EBADSIZE [38] *Size must be nonnegative*. The size of a marker or line must be greater or equal to zero. The current value of the *marker size* remains unchanged. Changing the value of the size argument to a non-negative value will correct this error.

EBADFABX [39] *Fill area index is invalid*. The fill area bundle is not defined. Corrective action involves changing the *index* argument to `fill_area_bundle_index`, or by defining the polymarker bundle index.

EPATARTL [40] *Pattern array too large*. The pattern array must contain less than 257 elements. The pattern is not entered into the pattern table. Corrective action involves designing a new pattern.

EPATSZTS [41] *Pattern size too small*. The pattern size must be at least two-by-two. The pattern is not entered into the pattern table. Corrective action could include designing a new pattern which includes several replications of the original pattern.

ESTYLLEZ [42] *Style (pattern or hatch) index is less than zero*. All indices in the pattern table must be positive. To fix this mistake, change the argument to the `pattern_index` or the `hatch_index` or the entries in the bundle table.

ENOPATNX [43] *Pattern table index not defined*. The argument to the `hatch_index` or `pattern_index` function or the entry bundle table should be reset to correspond to a defined value.

EPATITOL [44] *Pattern table index too large*. The *index* argument to `pattern_table` exceeded the bounds of the *pattern table*. The pattern is not entered into the *pattern table*. Redefining the pattern index to be between one and ten will eliminate the error.

EBADTXTX [45] *Text index is invalid*. The text bundle is not defined. Corrective action involves changing the *index* argument to `text_bundle_index`, or by defining the text bundle index.

EBDCHRIX [46] *Character index is undefined*. All other character indices besides 1 are undefined in SunCGI. The new *character index* is simply ignored. You are advised to ignore the `character_index` function entirely.

ETXTFLIN [47] *Text font is invalid*. The text fonts range from 1 to 6. All other integers do not correspond to actual fonts. Corrective action involves changing the argument to the `text_font_index` function or resetting the font index in the text bundle

ECEXFOOR [48] *Expansion factor is out of range*. The *character expansion factor* or the *character space expansion factor* would result in a character or a space which would exceed the bounds of the screen or would result in a character smaller than the limitations of the character drawing software. To eliminate this error, reset the offending value to within an acceptable range (0.1-2.0 are reasonable guidelines).

ECHHTLEZ [49] *Character height is less than or equal to zero*. The *character height* must be positive. Corrective action involves changing the argument to the *character height* function or the element of the text bundle.

ECHRUPVZ [50] *Length of character up vector or character base vector is zero*. Both the character up vector and the character base vector must be nonzero. Corrective action involves changing the arguments to the `character_orientation` function or the element of the text bundles.

ECOLRNGE [51] *RGB values must be between 0 and 255*. The red, green, and blue values are only defined between 0 and 255. The call to `color_table` which produced the error is ignored. Corrective action requires respecifying the values of the arguments to `color_table`.

D.6. Output Primitives (60-70)

ENMPTSTL [60] *Number of points is too large*. The number of points exceeds 255. Change the *n* element of the `Ccoorlist` structure to a value less than or equal to 255.

EPLMTWPT [61] *polylines must have at least two points*. Change the *n* element of the `Ccoorlist` structure to a value greater than

or equal to 2 and add the corresponding points to the *ptlist* element.

EPGMTHPT [62] *Polygons must have at least three points*. Change the *n* element of the `Ccoorlist` structure to a value greater than or equal to 3 and add the corresponding points to the *ptlist* element.

EGPLISFL [63] *Global polygon list is full*. The number of points on the *global polygon list* exceeds 256. The points which exceed 256 are ignored. This error can be corrected by inserting a call to `polygon` (which clears the *global polygon list* by displaying its contents) before the call to `partial_polygon` which caused the overflow.

EARCPNCI [64] *Arc points do not lie on circle*. The starting and ending points of either an open or close circular arc do not lie on the perimeter of the circle described by the arguments *c1* and *rad*. If this error occurs, the arc is not drawn. Corrective action may include determination of the endpoints with the application program (for example c2.x = rad*cos(start_angle);).

EARCPNEL [65] *Arc points do not lie on ellipse*. The starting and ending points of either an open or close elliptical arc do not lie on the perimeter of the ellipse described by the arguments *c1*, *c2*, and *c3*. If this error occurs, the arc is not drawn. Corrective action may include determination of the endpoints with the application program (see error 11).

ECELLATS [66] *Cell array dimensions dx,dy are too small*. The dimensions of the cell array are too small for a cell array element to be mapped onto one pixel of the view surface. The cell array is not drawn. This error depends on the physical size of the view surface as well as the limits of VDC space. Therefore, corrective action might require changing the size of the view surface, VDC space, or both.

ECELLPOS [67] *Cell array dimensions must be positive*. Negative cell array dimensions are not permitted. Corrective action requires changing the parameters to the *cell array* function.

ECELLTLS [68] Is not used.

EVALOVWS [69] *Value outside of view surface*. A coordinate of a pixel array is outside the physical range of the view surface. The pixel array is not drawn. Change the arguments to the `pixel_array` or `bitblt_source_array`

EPXNOTCR [70] *Pixrect not created*. One of the BitBlt functions required a user-defined *pixrect*, and that *pixrect* had not been created. Corrective action involves creating a *pixrect* in

your application program before calling the offending BitBlt function.

D.7. Input (80-97)

EINDNOEX [80] *Input device does not exist*. The input device specification (specified by the *devclass* and *devnum* arguments of most input functions) does not exist. Corrective action involves resetting the device specification to a valid device.

EINDINIT [81] *Input device not initialized*. A call to an input device function specified a device which was not initialized. Calls which generate this error have no effect. A call to `initialize_input_device` should be inserted before the call generating the error.

EINDALIN [82] *Input device already initialized*. An attempt to initialize a device which has previously been initialized. The parameters to the offending call to `initialize_input_device` are ignored. Removing the offending call to `initialize_input_device` will correct this error.

EINASAEX [83] *Association already exists*. An attempt is being made to bind the input device to a trigger to which it has been previously bound. The status of the input device trigger are unchanged. This error is purely informational and no corrective action is required.

EINAIIMP [84] *Association is impossible*. An attempt is being made to bind the input device to a trigger to which it cannot be bound. For example a IC_STRING device cannot be bound to a mouse button. To eliminate this error, change the arguments to the offending call of the `associate` function.

EINNTASD [85] *Association does not exist*. An attempt to set-up call an input function which specifies a device with no associated triggers was made. The offending call is ignored. Corrective action involves calling `associate` before the offending call is issued.

EINTRNEX [86] *Trigger does not exist*. An attempt was made to associate or inquire about a trigger which has a number less than one or greater than five. The offending call is ignored. To eliminate the error, change the trigger number.

EINNECHO [87] *Input device does not echo*. CHOICE devices do not support echo. Corrective action requires removing the call to `echo_on` from the application program.

EINECHON [88] *Echo already on*. A call to `echo_on` has been made to a device whose echoing ability has already been activated. To stop generation of the error either remove the offending call or change the arguments to specify a device whose

echo is currently off.

EINEINCP [89] *Echo incompatible with existing echos*. Although SunCGI can support certain combinations of echos (such as IC_STRING and IC_LOCATOR), not all combinations are supported. The easiest remedy is to remove the most recent call to `echo_on` from the application program.

EINERVWS [90] *Echoregion larger than view surface*. Error 91 is generated when the rectangle defined by the *echoregion* argument exceeds the limits of VDC space. To eliminate this error, change the values to the *echoregion* argument to be within the confines of VDC space.

EINETNSU [91] *Echo type not supported*. All devices except the IC_STROKE device only support one type of echo. Therefore, assigning a value to *echotype* other than zero or one will produce an error for any device except IC_STROKE. Corrective action involves changing the value of the *echotype* argument.

EINENOTO [92] *Echo not on*. The device echoing has not been turned on. Either remove the call to `echo_off`, turn the echo on, or change the device specification.

EIAEVNEN [93] *Events already enabled*. Events have already been enabled for the specified device. The solution is to remove the offending call to `enable_events`.

EINEVNEN [94] *Events not enabled*. Events have not been enabled for the specified device. The solution is to include a call to `enable_events` before a call to the `await_event`, `sample_event`, or `request_event` function is made with the specified device as input parameter.

EBADDATA [95] *Contents of input data record are invalid*. The *value* argument of `initialize_lid` function is out of range or is the wrong type. The solution is to change the contents *value* argument.

ESTRSIZE [96] *Length of initial string is greater than the implementation defined maximum*. The initial string in the value argument is greater than 80 characters. Shorten the string.

EINQOVFL [97] *Input queue has overflowed*. The *event queue* can no longer record input events. Solutions include flushing the *event queue* or dequeueing events with the `await_event`, `sample_event`, or `request_event` function.

D.8. Implementation Dependent (110-112)

`EMEMSPAC [110]` *Space allocation has failed*. A function which was supposed to work has failed. The only action which you can take is to eliminate other processes which may be using memory. If you have eliminated all other processes, and this error is still generated, please contact SUN Microsystems.

`ENOTCSTD [111]` *Function or argument not compatible with standard CGI*. A function call is not supported by the CGI library.

`ENOTCCPW [112]` *Function or argument not compatible with CGIPW mode*. A function call is not supported by the `cgipw` library.

D.9. Possible Causes of Visual Errors

Table D-1 *Possible Causes of Visual Errors*

Behavior	*Possible Cause*
Segmentation fault for `open_vws`	*devdd* argument for `open_vws` is declared as a pointer (the address of *devdd* should be passed).
No primitives displayed	View surface not initialized. View surface not active. VDC to device coordinate mapping makes objects too small. Clipping rectangle is too small and clipping is ON. Perimeter visibility is set to OFF and interior style is set to HOLLOW. *line colo*. or *fill colo*. is set to background color.
Primitives displayed on undesired view surfaces	Undesired view surfaces have not been deactivated.
Segmentation fault for inquiry functions	passing variable instead of address (&) of variable.

Table D-2 *Primitive-Specific Errors*

Behavior	*Possible Cause*
Polylines or polymarkers aren't displayed.	Width or size is zero. Color is the same as background.
Polygon borders aren't displayed.	Width is zero. Color is the same as background. Perimeter visibility is set to OFF.
Circles aren't displayed.	Width or size is zero. Color is the same as background.
Ellipses aren't displayed.	Width or size is zero. Color is the same as background.
Text isn't displayed.	Width or size is zero. Color is the same as background. character height is too small. coordinates are outside the range of VDC space or the clipping rectangle.
Cell arrays aren't displayed.	*dx* or *dy* arguments are too small. Color is the same as background.
Cell arrays aren't displayed on all active view surfaces.	Mapping from cell size to view surface for smaller view surfaces is too small.
Pixel arrays aren't displayed.	Location is outside of view surface or clipping rectangle. Color is the same as background.
BitBlts aren't displayed.	Width or size is zero. Color is the same as background.

Table D-3 *Attribute Errors*

Behavior	*Possible Cause*
Attribute setting has no effect	attribute ASF is set to BUNDLED.
Text attributes have no effect	text precision is set to CHARACTER. attribute ASF is set to BUNDLED.
PATTERN fill is the same as HATCH	*pattern index*. and hatch index are identical *pattern size*. is too small
PATTERN fill is different on different view surfaces.	View surfaces are of different size.

Table D-4 *Input-specific Errors*

Behavior	*Possible Cause*
Input device does not report	device not initialized
Input device does not echo	echo not initialized
Input device does not echo on whole view surface	echo region not set to whole view surface.

E

Sample Programs

E

Sample Programs

E.1. Martini Glass

The following program draws a martini glass. The program exits after 10 seconds.

```
#include <cgidefs.h>

Ccoorlist martinilist;
Ccoor glass_coords[10] = { 0,0,
                           -10,0,
                           -1,1,
                           -1,20,
                           -15,35,
                           15,35,
                           1,20,
                           1,1,
                           10,0,
                           0,0 };
Ccoor water_coords[2] = { -12,33,
                          12,33 };
Ccoor vpll = { -50,-10 };
Ccoor vpur = { 50,80 };

main()
{
    Cvwsurf device;
    Cint name;

    NORMAL_VWSURF(device, PIXWINDD);

    open_cgi();
    open_vws(&name, &device);
    vdc_extent(&vpll, &vpur);

    martinilist.ptlist = glass_coords;
    martinilist.n = 10;
    polyline(&martinilist);
    martinilist.ptlist = water_coords;
    martinilist.n = 2;
    polyline(&martinilist);

    sleep(10);
    close_vws(name);
    close_cgi();
}
```

Figure E-1 *Martini Glass Example Program*

E.2. Tracking Box

The following program demonstrates the use of the CGI input functions. A square is displayed on the screen and moved with the mouse. The program exits if the mouse is still for five seconds.

```
#include <cgidefs.h>
#define DEVNUM  1           /* device number */
#define MOUSE_POSITION  5   /* trigger number */
#define TIMEOUT (5 * 1000 * 1000)   /* timeout in microseconds */

Ccoor ulc = {1000, 2000};
Ccoor lrc = {2000, 1000};

main()
{
    Cint name;
    Cvwsurf device;
    Cawresult stat;
    Cinrep sample;  /* device measure value */
    Ccoor samp;     /* LOCATOR's x,y position */
    Cint trigger;   /* trigger number */

    NORMAL_VWSURF(device, PIXWINDD);
    sample.xypt = &samp;
    samp.x = 0;
    samp.y = 27000;

    open_cgi();
    open_vws(&name, &device);
    set_global_drawing_mode(XOR);
    initialize_lid(IC_LOCATOR, DEVNUM, &sample);
    associate(MOUSE_POSITION, IC_LOCATOR, DEVNUM);
    rectangle(&lrc, &ulc);  /* draw first rectangle */
        /* wait TIMEOUT micro-seconds for input and check the status */
    while (request_input(IC_LOCATOR, DEVNUM, TIMEOUT,
        &stat, &sample, &trigger), (stat == VALID_DATA)) {
        if ((sample.xypt->x != ulc.x) || (sample.xypt->y != lrc.y) ) {
            rectangle(&lrc, &ulc);
            lrc.y = sample.xypt->y; /* move to new location */
            lrc.x = (sample.xypt->x + 1000);
            ulc.x = sample.xypt->x;
            ulc.y = (sample.xypt->y + 1000);
            rectangle(&lrc, &ulc);
        }
    }
    dissociate(MOUSE_POSITION, IC_LOCATOR, DEVNUM);
    release_input_device(IC_LOCATOR, DEVNUM);
    close_vws(name);
    close_cgi();
}
```

Figure E-2 *Tracking Box Example Program*

F

Using SunCGI and Pixwins

F

Using SunCGI and Pixwins

The CGI standard does not provide facilities for dealing with multiple overlapping windows. An application program can use SunCGI and Pixwins features through the `cgipw` functions. These functions combine the richness of CGI's primitives with the ability of Pixwins to manage multiple (potentially overlapping) windows.

This appendix assumes familiarity with both SunCGI and Pixwins. See *SunView Programmer's Guide* for more information on Pixwins. An example program is included at the end of this appendix in Figure F-1.

If you decide to use CGI and Pixwins, you *may not* use the standard SunCGI calls. Instead you must use `cgipw` calls. For example, `cgipw_polyline` replaces `polyline`. The first argument of each `cgipw` function is a pixwin descriptor of type `Ccgiwin`. The file `<cgipw.h>` must be included in the `cgipw` application program instead of `<cgidefs.h>`.

F.1. `cgipw` Functions

The four functions `open_pw_cgi`, `open_cgi_pw`, `close_cgi_pw` and `close_pw_cgi` are necessary for managing the SunCGI – Pixwins interface.

Open Pixwin CGI

```
Cerror open_pw_cgi()
```

`open_pw_cgi` initializes CGI by setting the attributes to the default values and setting the VDC to device coordinate mapping to 1:1. Therefore, all input and output primitives will use device coordinates. The origin of the device coordinates is in the upper left-hand corner instead of the lower left-hand corner. The entire window is used, not just a square region within it. No standard errors are specified for `open_pw_cgi`. If `open_pw_cgi` returns a nonzero result, then the initialization failed. `open_pw_cgi` corresponds to `open_cgi`.

Open a CGI Pixwin

```
Cerror open_cgi_pw(pw, desc, name)
struct pixwin *pw; /* pixwin */
Ccgiwin *desc; /* CGI pixwin descriptor */
Cint *name;
```

`open_cgi_pw` informs CGI of the pixwin pointed to by *pw*. Calls to CGI primitives may then reference this pixwin. However, CGI does not guarantee that a pixwin exists or is in any other way properly initialized. *desc* is a pointer to a CGI pixwin descriptor allocated by the application program and defined by `open_cgi_pw`. It will be used as the first argument to `cgipw` functions. Calls

may also be made to any pixwin function (see example program). Multiple calls to `open_cgi_pw` with pointers to different `Ccgiwin` structures will allow primitives to be displayed on multiple view surfaces by repeating calls to `cgipw` functions with different `Ccgiwin` descriptors. Attributes are local to the pixwin associated with the CGI descriptor passed to the `cgipw` attribute functions. `open_cgi_pw` corresponds to `open_vws`. `open_pw_cgi` must be called prior to `open_cgi_pw`; otherwise, error 111 is returned. Other errors (as with `open_vws` may also be detected.

Errors

`ENOTOPOP` [5]	CGI not in proper state CGI shall be either in state CGOP, VSOP, or VSAC.
`ENOWSTYP` [11]	Specified view surface type does not exist.
`EMAXVSOP` [12]	Maximum number of view surfaces already open.
`EMEMSPAC` [110]	Space allocation has failed.

Close a CGI Pixwin

```
Cerror close_cgi_pw(desc)
Ccgiwin *desc; /* CGI pixwin descriptor */
```

`close_cgi_pw` takes the CGI pixwin descriptor *desc* as an argument and removes it from the list of pixwins that CGI writes to. The pixwin is *not* closed. `close_cgi_pw` corresponds to `close_vws`, and may return any of the errors `close_vws` detects (except 112).

Errors

`ENOTOPOP` [5]	CGI not in proper state CGI shall be either in state CGOP, VSOP, or VSAC.
`EVSIDINV` [10]	Specified view surface name is invalid.
`EVSNOTOP` [13]	Specified view surface not open.

Close Pixwin CGI

```
Cerror close_pw_cgi()
```

`close_pw_cgi` takes care of leaving CGI in an orderly state. This function should be called before exiting the application program. `close_pw_cgi` corresponds to `close_cgi`.

Errors

`ENOTOPOP` [5]	CGI not in proper state CGI should be in state CGOP, VSOP, or VSAC.

F.2. Using `cgipw`

After calling the two initialization functions (`open_pw_cgi` and `open_cgi_pw`) the application program may call functions from both the Pixwins and `cgipw` libraries. Figure F-1 contains an example program that uses `cgipw` functions.

Since `cgipw` functions use a 1:1 mapping from VDC to device coordinates, attributes in VDC units (such as *pattern size* and *character height*) will be huge unless they are reset. And because the `cgipw` origin is the device coordinate origin, the upper left-hand corner, attributes with direction or position (e.g., *pattern reference point* and *character orientation*) have their meaning reversed in

the *y* dimension.

Most cgipw functions do not print error messages even if the error warning mask is INTERRUPT or POLL. They all return error codes which may be tested. The application program should not use *both* SunCGI and window system input functions, since both SunCGI and the window system share a common event queue. For example, events handled by a SunCGI function will not be handled by a window system call after the SunCGI call.

A list of the cgipw functions and their corresponding SunCGI functions is given in Table F-1 below. If a function is not included in this table, then use the normal SunCGI function except as described below in Table F-2. Most of the functions listed below are output and attribute functions; however, the tracking functions are listed so that you can control which surfaces input devices echo on. The arguments of the cgipw functions are the same as those of the SunCGI functions except that the first argument is always a *desc* argument of type Ccgiwin. *desc* is a pointer to a pixwin descriptor filled in by the open_cgi_pw function.

F.3. cgipw Functions

Table F-1 contains a list of functions available in cgipw mode. SunCGI functions incompatible with cgipw mode are given in Table F-2. partial_polygon may be used with cgipw_polygon, but the *global polygon list* is freed after use by cgipw_polygon, so calls to partial_polygon must be repeated prior to use of cgipw_polygon on another view surface.

Table F-1 *List of* cgipw *Functions*

SunCGI Function Name	cgipw *Function Name*
append_text(flag, tstring)	cgipw_append_text(desc, flag, tstring)
cell_array(p, q, r, dx, dy, colorind)	cgipw_cell_array(desc, p, q, r, dx, dy, colorind)
character_expansion_factor(sfac)	cgipw_character_expansion_factor(desc, sfac)
character_height(height)	cgipw_character_height(desc, height)
character_orientation(xup, yup, xbase, ybase)	cgipw_character_orientation(desc, xup, yup, xbase, ybase)
character_path(path)	cgipw_character_path(desc, path)
character_set_index(index)	cgipw_character_set_index(desc, index)
character_spacing(spcratio)	cgipw_character_spacing(desc, spcratio)
circle(c1, rad)	cgipw_circle(desc, c1, rad)
circular_arc_3pt(c1, c2, c3)	cgipw_circular_arc_3pt(desc, c1, c2, c3)
circular_arc_3pt_close(c1, c2, c3, close)	cgipw_circular_arc_3pt_close(desc, c1, c2, c3, close)
circular_arc_center(c1, c2x, c2y, c3x, c3y, rad)	cgipw_circular_arc_center(desc, c1, c2x, c2y, c3x, c3y, rad)
circular_arc_center_close(c1, c2x, c2y, c3x, c3y, rad, close)	cgipw_circular_arc_center_close(desc, c1, c2x, c2y, c3x, c3y, rad, close)
color_table(istart, clist)	cgipw_color_table(desc, istart, clist)
define_bundle_index(index)	cgipw_define_bundle_index(desc, index)
disjoint_polyline(polycoors)	cgipw_disjoint_polyline(desc, polycoors)
ellipse(c1, majx, miny)	cgipw_ellipse(desc, c1, majx, miny)

Table F-1 *List of* cgipw *Functions— Continued*

SunCGI Function Name	cgipw *Function Name*
elliptical_arc(c1, sx, sy, ex, ey, majx, miny)	cgipw_elliptical_arc(desc, c1, sx, sy, ex, ey, majx, miny)
elliptical_arc_close(c1, sx, sy, ex, ey, majx, miny, close)	cgipw_elliptical_arc_close(desc, c1, sx, sy, ex, ey, majx, miny, close)
fill_area_bundle_index(index)	cgipw_fill_area_bundle_index(desc, index)
fill_color(color)	cgipw_fill_color(desc, color)
fixed_font(index)	cgipw_fixed_font(desc, index)
hatch_index(index)	cgipw_hatch_index(desc, index);
inquire_aspect_source_flags()	cgipw_inquire_aspect_source_flags(desc);
inquire_drawing_mode(visibility, source, destination, combination)	cgipw_inquire_drawing_mode(desc, visibility, source, destination, combination)
inquire_fill_area_attributes()	cgipw_inquire_fill_area_attributes(desc);
inquire_line_attributes()	cgipw_inquire_line_attributes(desc);
inquire_marker_attributes()	cgipw_inquire_marker_attributes(desc);
inquire_pattern_attributes()	cgipw_inquire_pattern_attributes(desc);
inquire_pixel_array(p, m, n, colorind)	cgipw_inquire_pixel_array(desc, p, m, n, colorind)
inquire_text_attributes()	cgipw_inquire_text_attributes(desc);
inquire_text_extent(tstring, nextchar, concat, lleft, uleft, uright)	cgipw_inquire_text_extent(desc, tstring, nextchar, concat, lleft, uleft, uright)
interior_style(istyle, perimvis)	cgipw_interior_style(desc, istyle, perimvis)
line_color(index)	cgipw_line_color(desc, index)
line_endstyle(ttyp)	cgipw_line_endstyle(desc, ttyp)
line_type(ttyp)	cgipw_line_type(desc, ttyp)
line_width(index)	cgipw_line_width(desc, index)
line_width_specification_mode(mode)	cgipw_line_width_specification_mode(desc, mode)
marker_color(index)	cgipw_marker_color(desc, index)
marker_size(index)	cgipw_marker_size(desc, index)
marker_size_specification_mode(mode)	cgipw_marker_size_specification_mode(desc, mode)
marker_type(ttyp)	cgipw_marker_type(desc, ttyp)
pattern_index(index)	cgipw_pattern_index(desc, index);
pattern_reference_point(open)	cgipw_pattern_reference_point(desc, open)
pattern_size(dx, dy)	cgipw_pattern_size(desc, dx, dy)
perimeter_color(index)	cgipw_perimeter_color(desc, index)
perimeter_type(ttyp)	cgipw_perimeter_type(desc, ttyp)
perimeter_width(width)	cgipw_perimeter_width(desc, width)
perimeter_width_specification_mode(mode)	cgipw_perimeter_width_specification_mode(desc, mode)
pixel_array(pcell, m, n, colorind)	cgipw_pixel_array(desc, pcell, m, n, colorind)
polygon(polycoors)	cgipw_polygon(desc, polycoors)
polyline(polycoors)	cgipw_polyline(desc, polycoors)
polyline_bundle_index(index)	cgipw_polyline_bundle_index(desc, index)
polymarker(polycoors)	cgipw_polymarker(desc, polycoors)
polymarker_bundle_Index(index)	cgipw_polymarker_bundle_Index(desc, index)
rectangle(lrc, ulc)	cgipw_rectangle(desc, lrc, ulc)
set_aspect_source_flags(flags)	cgipw_set_aspect_source_flags(desc, flags)
text(c1, tstring)	cgipw_text(desc, c1, tstring)

Table F-1 *List of* cgipw *Functions— Continued*

SunCGI Function Name	cgipw *Function Name*
text_alignment(halign, valign, hcalind, vcalind)	cgipw_text_alignment(desc, halign, valign, hcalind, vcalind)
text_bundle_index(index)	cgipw_text_bundle_index(desc, index)
text_color(index)	cgipw_text_color(desc, index)
text_font_index(index)	cgipw_text_font_index(desc, index)
text_precision(ttyp)	cgipw_text_precision(desc, ttyp)
vdm_text(c1, flag, tstring)	cgipw_vdm_text(desc, c1, flag, tstring)

Table F-2 SunCGI *Functions not Compatible with* cgipw *Mode*

Function	*Discussion*
clear_control	All clear extents are identical
clip_indicator	when *cflag* is CLIP_RECTANGLE
clip_rectangle	Instead, use pw_region prior to open_cgi_pw
close_cgi	Use close_pw_cgi
close_vws	Use close_cgi_pw
device_viewport	use pw_region prior to open_cgi_pw
open_cgi	Use open_pw_cgi
open_vws	Use open_cgi_pw
partial_polygon	*global polygon list* is freed after cgipw_polygon
vdc_extent	cgipw's VDC space is identical to screen space

F.4. Example Program

Figure F-1 contains an example program that uses cgipw functions. This example uses retained pixwins to ease redisplay after window obstruction (see Section 2.3). This makes the program slower during image generation, because it writes both on the screen and onto a copy retained in memory.

```
#include <cgipw.h>
#include <suntool/gfxsw.h>

struct pixwin *mypw;
struct gfxsubwindow *mine;

main()
{
    Ccgiwin vpw;
    Ccoor bottom;
    Ccoor top;
    int name;
    int op;

    mine = gfxsw_init(0, 0);
    gfxsw_getretained(mine);
    mypw = mine->gfx_pixwin;
    pw_writebackground(mypw, 0, 0,
        mypw->pw_prretained->pr_size.x,
        mypw->pw_prretained->pr_size.y, PIX_CLR);

    open_pw_cgi();
    open_cgi_pw(mypw, &vpw, &name);
    op = PIX_COLOR(1) | PIX_SRC;
    pw_write(mypw, 0, 0, 100, 100, op, 0, 0, 0);
    bottom.x = 300;
    bottom.y = 100;
    top.x = 200;
    top.y = 0;
    cgipw_interior_style(&vpw, SOLIDI, ON);
    cgipw_rectangle(&vpw, &bottom, &top);
    sleep(10);

    close_cgi_pw(&vpw);
    close_pw_cgi();
}
```

Figure F-1 *Example* `cgipw` *Program*

G

Using SunCGI with Fortran Programs

G

Using SunCGI with Fortran Programs

All functions provided in SunCGI may be called from FORTRAN programs by linking them with the `libcgi77.a` library. This is done by using the *f77* compiler with a command line like:

```
% f77 -o box box.f -lcgi77 -lcgi -lsunwindow -lpixrect -lm
```

where `box.f` is the FORTRAN source program. Note that `libcgi.a` must be linked with the program (the `-lcgi` option), and `libcgi77.a` must precede it (the `-lcgi77` option).

Defined constants may be referenced in source programs by including `cgidefs77.h`. In a FORTRAN program, this must be done via a source statement like:

```
include 'cgidefs77.h'
```

This include statement must be in each FORTRAN program unit which uses the defined constants, not just once in each source program file.

In the Sun release of FORTRAN, names are restricted to sixteen characters in length and may not contain the underline character. For this reason, FORTRAN programs must use abbreviated names to call the corresponding SunCGI functions. The correspondence between the full SunCGI names and the FORTRAN names appears later in this appendix. In addition, FORTRAN declarations for all SunCGI functions appear at the end of this appendix.

G.1. Programming Tips

- The abbreviated names of the SunCGI functions are less readable than the full length names because the underline character cannot be used in the FORTRAN names. However, since FORTRAN doesn't distinguish between upper-case and lower-case letters in names, upper-case characters can be used to improve readability. There is an example of this later in this appendix.
- Character strings passed from FORTRAN programs to SunCGI cannot be longer than 256 characters.
- Pointers returned by C functions are handled in FORTRAN as `integer*4` values, and exist solely to be passed to other Sun graphics functions.
- FORTRAN passes all arguments by reference. Although some SunCGI functions receive arguments by value, the FORTRAN programmer need not worry

about this. The interface routines in `/usr/lib/libcgi77.a` handle this situation correctly. When in doubt, look at the FORTRAN declarations for SunCGI functions at the end of this appendix.

- Some SunCGI functions have structures as arguments or return values. These are handled in FORTRAN by unbundling the structures into separate arguments. In general, these will be in the same order, and have the same names, as the members of the C structures. One exception is the `Ccoorlist` structure, which is replaced in FORTRAN with an array of *x*'s, and one of *y*'s, rather than an array of *x*-*y* pairs. You may need to consult both the C and FORTRAN documentation to determine which FORTRAN arguments are input values, and which are output.
- Since FORTRAN does not distinguish between upper-case letters and lower-case letters in identifiers, any FORTRAN program unit which includes the `cgidefs77.h` header file cannot use the same spelling as any constant defined in that header file, regardless of case.
- The function `cfqoutcap` returns the FORTRAN binding names of the output capabilities, rather than the C bindings. This is an exception to the rule that the FORTRAN library provides a transparent interface to the C functions.

G.2. Example Program

This example is the FORTRAN equivalent of the very simple program for drawing a martini glass.

```
      program test

      parameter (ibignum=256)

      integer name
      character screenname* (ibignum)
      integer screenlen
      character windowname* (ibignum)
      integer windowlen
      integer windowfd
      integer retained
      integer dd
      integer cmapsize
      character cmapname* (ibignum)
      integer cmaplen
      integer flags
      character ptr* (ibignum)
      integer noargs
c         coordinates of glass
      integer xc(10),yc(10),n
c         coordinates of waterline.
      integer xc2(2),yc2(2)
      data xc /0,-10,-1,-1,-15,15,1,1,10,0 /
      data yc /0,0,1,20,35,35,20,1,0,0 /
      data xc2 /-12,12/
      data yc2 /33,33/


c         open cgi
      call cfopencgi()
c         open a pixwin
      dd = 4
      call cfopenvws(name,screenname,screenlen,windowname,
     +  windowlen,windowfd,retained,dd,cmapsize,
     +  cmapname,cmaplen,flags,ptr,noargs)
c         reset VDC space
      call cfvdcext(-50,-10,50,80)
c         draw martini glass and waterline
      n = 10
      call cfpolyline(xc,yc,n)
      n = 2
      call cfpolyline(xc2,yc2,n)
c         sleep for 10 seconds
      call sleep(10)
c         close and exit
      call cfclosecgi()
      call exit()
      end
```

Figure G-1 *Example FORTRAN Program*

G.3. FORTRAN Interfaces to SunCGI

Note: Although all SunCGI procedures are declared here as functions, each may also be called as a subroutine if the user does not want to check the returned value.

Table G-1 *SunCGI Fortran Binding – Part I*

CGI Specification Name	***Fortran Binding***
Activate View Surface (SunCGI Extension)	`integer function cfactvws(name)` `integer name`

Append Text	`integer function cfaptext(flag, string)`
	`integer flag`
	`character*(*) string`

Associate	`integer function cfassoc(trigger, devclass, devnum)`
	`integer trigger`
	`integer devclass`
	`integer devnum`

Await Event	`integer function cfawaitev(timeout, valid, devclass,`
	`1 devnum, x, y, xlist, ylist, n, val, choice, string,`
	`2 segid, pickid, message_link, replost, time_stamp,`
	`3 qstat, overflow)`
	`integer timeout`
	`integer valid`
	`integer devclass`
	`integer devnum`
	`integer x, y`
	`integer xlist(*)`
	`integer ylist(*)`
	`integer n`
	`real val`
	`integer choice`
	`character*(*) string`
	`integer segid`
	`integer pickid`
	`integer message_link`
	`integer replost`
	`integer time_stamp`
	`integer qstat`
	`integer overflow`

Table G-1 *SunCGI Fortran Binding – Part I— Continued*

CGI Specification Name	***Fortran Binding***

BitBlt Pattern Array	`integer function cfbtblpatarr(pixpat, px, py, pixtarget,`
	`1 rx, ry, ox, oy, dx, dy, name)`
	`integer pixpat`
	`integer px, py`
	`integer pixtarget`
	`integer rx, ry`
	`integer ox, oy`
	`integer dx, dy`
	`integer name`

BitBlt Patterned Source Array	`integer function cfbtblpatsouarr(pixpat, px, py, pixsource,`
	`1 sx, sy, pixtarget, rx, ry, ox, oy, dx, dy, name)`
	`integer pixpat`
	`integer px, py`
	`integer pixsource`
	`integer sx, sy`
	`integer pixtarget`
	`integer rx, ry`
	`integer ox, oy`
	`integer dx, dy`
	`integer name`

BitBlt Source Array	`integer function cfbtblsouarr(bitsource, xo, yo, xe, ye,`
	`1 bittarget, xt, yt, name)`
	`integer*4 bitsource, bittarget`
	`integer xo, yo, xe, ye, xt, yt`
	`integer name`

Cell Array	`integer function cfcellarr(px, qx, rx, py, qy, ry,`
	`1 dx, dy, colorind)`
	`integer px, py`
	`integer qx, qy`
	`integer rx, ry`
	`integer dx, dy`
	`integer colorind(*)`

Character Expansion Factor	`integer function cfcharexpfac(efac)` `real efac`
Character Height	`integer function cfcharheight(height)` `integer height`
Character Orientation	`integer function cfcharorient(bx, by, dx, dy)` `real bx, by, dx, dy`
Character Path	`integer function cfcharpath(path)` `integer path`
Character Set Index	`integer function cfcharsetix(index)` `integer index`

Table G-1 *SunCGI Fortran Binding – Part I— Continued*

CGI Specification Name	***Fortran Binding***
Character Spacing	`integer function cfcharspacing(efac)` `real efac`

Circle	`integer function cfcircle(x, y, rad)`
	`integer x`
	`integer y`
	`integer rad`

Circular Arc 3pt Close	`integer function cfcircarcthreecl(c1x, c1y, c2x, c2y,`
	`1 c3x, c3y, close)`
	`integer c1x, c1y, c2x, c2y, c3x, c3y`
	`integer close`

Circular Arc 3pt	`integer function cfcircarcthree(c1x, c1y, c2x, c2y,`
	`1 c3x, c3y)`
	`integer c1x, c1y, c2x, c2y, c3x, c3y`

Circular Arc Center Close	`integer function cfcircarccentcl(c1x, c1y, c2x, c2y,`
	`1 c3x, c3y, rad, close)`
	`integer c1x, c1y, c2x, c2y, c3x, c3y`
	`integer rad`
	`integer close`

Circular Arc Center	`integer function cfcircarccent(c1x, c1y, c2x, c2y, c3x,`
	`1 c3y, rad)`
	`integer c1x, c1y, c2x, c2y, c3x, c3y`
	`integer rad`

Clear Control	`integer function cfclrcont(soft, hard, intern, extent)`
	`integer soft, hard`
	`integer intern`
	`integer extent`

Clear View Surface	`integer function cfclrvws(name, defflag, color)`
	`integer name`
	`integer defflag`
	`integer color`

Clip Indicator	`integer function cfclipind(flag)` `integer flag`
Clip Rectangle	`integer function cfcliprect(xmin, xmax, ymin, ymax)` `integer xmin, xmax, ymin, ymax`
Close CGI (SunCGI Extension)	`integer function cfclosecgi()`
Close View Surface (SunCGI Extension)	`integer function cfclosevws(name)` `integer name`

Table G-2 *SunCGI Fortran Binding – Part II*

CGI Specification Name	***Fortran Binding***

Color Table	`integer function cfcotable(istart, ra, ga, ba, n)`
	`integer istart`
	`integer ra(*), ga(*), ba(*)`
	`integer n`

Deactivate View Surface (SunCGI Extension)	`integer function cfdeactvws(name)` `integer name`

Define Bundle Index (SunCGI Extension)	`integer function cfdefbundix(index, linetype, linewidth,`
	`1 linecolor, marktype, marksize, markcolor, intstyle,`
	`2 batchindex, pattindex, fillcolor, perimtype,`
	`3 perimwidth, perimcolor, t3extfont, textprec,`
	`4 charexpand, charspace, textcolor)`
	`integer index`
	`integer linetype`
	`real linewidth`
	`integer linecolor`
	`integer marktype`
	`real marksize`
	`integer markcolor`
	`integer intstyle`
	`integer batchindex`
	`integer pattindex`
	`integer fillcolor`
	`integer perimtype`
	`real perimwidth`
	`integer perimcolor`
	`integer t3extfont`
	`integer textprec`
	`real charexpand`
	`real charspace`
	`integer textcolor`

Device Viewport	`integer function cfdevvpt(name, xbot, ybot, xtop, ytop)`
	`integer name`
	`integer xbot, ybot, xtop, ytop`

Disable Events	`integer function cfdaevents(devclass, devnum)`
	`integer devclass`
	`integer devnum`

Disjoint Polyline	`integer function cfdpolyline(xcoors, ycoors, n)`
	`integer xcoors(*)`
	`integer ycoors(*)`
	`integer n`

Table G-2 *SunCGI Fortran Binding – Part II— Continued*

CGI Specification Name	***Fortran Binding***

Dissociate	`integer function cfdissoc(trigger, devclass, devnum)`
	`integer trigger`
	`integer devclass`
	`integer devnum`

Ellipse	`integer function cfellipse(x, y, majx, miny)`
	`integer x, y`
	`integer majx, miny`

Elliptical Arc Close	`integer function cfelliparccl(x, y, sx, sy, ex, ey,`
	`1 majx, miny, close)`
	`integer x, y`
	`integer sx, sy`
	`integer ex, ey`
	`integer majx, miny`
	`integer close`

Elliptical Arc	`integer function cfelliparc(x, y, sx, sy, ex, ey, majx,`
	`1 miny)`
	`integer x, y`
	`integer sx, sy`
	`integer ex, ey`
	`integer majx, miny`

Enable Events	`integer function cfenevents(devclass, devnum)`
	`integer devclass`
	`integer devnum`

Fill Area Bundle Index	`integer function cfflareabundix(index)` `integer index`
Fill Color	`integer function cfflcolor(color)` `integer color`
Fixed Font *(SunCGI Extension)*	`integer function cffixedfont(index)` `integer index`
Flush Event Queue	`integer function cfflusheventqu()`

Table G-2 *SunCGI Fortran Binding – Part II— Continued*

CGI Specification Name	***Fortran Binding***

Get Last Requested Input	`integer function cfgetlastreqinp(devclass, devnum, valid,`
	`1 x, y, xlist, ylist, n, val, choice, string, segid,`
	`2 pickid)`
	`integer devclass`
	`integer devnum`
	`integer valid`
	`integer x, y`
	`integer xlist(*)`
	`integer ylist(*)`
	`integer n`
	`real val`
	`integer choice`
	`character*(*) string`
	`integer segid`
	`integer pickid`

Hard Reset	`integer function cfhardrst()`
Hatch Index	`integer function cfhatchix(index)` `integer index`

Initialize LID	`integer function cfinitlid(devclass, devnum, x, y, xlist,`
	`1 ylist, n, val, choice, string, segid, pickid)`
	`integer devclass`
	`integer devnum`
	`integer x, y`
	`integer xlist(*)`
	`integer ylist(*)`
	`integer n`
	`real val`
	`integer choice`
	`character*(*) string`
	`integer segid`
	`integer pickid`

Initiate Request	`integer function cfinitreq(devclass, devnum)`
	`integer devclass`
	`integer devnum`

Inquire Aspect Source Flags	`integer function cfqasfs(n, num, vals)`
	`integer n`
	`integer num(*)`
	`integer vals(*)`

Table G-2 *SunCGI Fortran Binding – Part II— Continued*

CGI Specification Name	***Fortran Binding***

Inquire BitBlt Alignments	`integer function cfqbtbltalign(base, width, px, py,`
	`1 maxpx, maxpy, name)`
	`integer base`
	`integer width`
	`integer px`
	`integer py`
	`integer maxpx`
	`integer maxpy`
	`integer name`

Inquire Cell Array	`integer function cfqcellarr(name, px, qx, rx, py, qy,`
	`1 ry, dx, dy, colorind)`
	`integer name`
	`integer px, py`
	`integer qx, qy`
	`integer rx, ry`
	`integer dx, dy`
	`integer colorind(*)`

Inquire Device Bitmap	`integer function cfqdevbtmp(name, map)`
	`integer name`
	`integer*4 map`

Inquire Device Class	`integer function cfqdevclass(output, input)` `integer output, input`

Table G-3 *SunCGI Fortran Binding – Part III*

CGI Specification Name	***Fortran Binding***

Inquire Device Identification	`integer function cfqdevid(name, devid)`
	`integer name`
	`character*(*) devid`

Inquire Drawing Mode	`integer function cfqdrawmode(visibility, source,`
	`1 destination, combination)`
	`integer visibility`
	`integer source`
	`integer destination`
	`integer combination`

Inquire Event Queue State	`integer function cfqevque(qstate, qoflow)`
	`integer qstate`
	`integer qoflow`

Table G-3 *SunCGI Fortran Binding – Part III— Continued*

CGI Specification Name	***Fortran Binding***

Inquire Fill Area Attributes	`integer function cfqflareaatts(style, vis, color, hindex,`
	`1 pindex, bindex, pstyle, pwidth, pcolor)`
	`integer style, vis, color`
	`integer hindex, pindex, bindex`
	`integer pstyle`
	`real pwidth`
	`integer pcolor`

Inquire Input Capabilities	`integer function cfqinpcaps(valid, numloc, numval, numstrk,`
	`1 numchoice, numstr, numtrig, evqueue, asynch, coordmap,`
	`2 echo, tracking, prompt, acknowledgement, trigman)`
	`integer valid`
	`integer numloc`
	`integer numval`
	`integer numstrk`
	`integer numchoice`
	`integer numstr`
	`integer numtrig`
	`integer evqueue`
	`integer asynch`
	`integer coordmap`
	`integer echo`
	`integer tracking`
	`integer prompt`
	`integer acknowledgement`
	`integer trigman`

Table G-3 *SunCGI Fortran Binding – Part III— Continued*

CGI Specification Name	***Fortran Binding***

Inquire LID State List	`integer function cfqlidstatelis(devclass, devnum, valid,`
	`1 state, prompt, acknowledgement, x, y, xlist, ylist, n,`
	`2 val, choice, string, segid, pickid, n, triggers,`
	`3 echotype, echosta, echodat)`
	`integer devclass`
	`integer devnum`
	`integer valid`
	`integer state`
	`integer prompt`
	`integer acknowledgement`
	`integer x`
	`integer y`
	`integer xlist(*)`
	`integer ylist(*)`
	`integer n`
	`real val`
	`integer choice`
	`character*(*) string`
	`integer segid`
	`integer pickid`
	`integer n`
	`integer triggers(*)`
	`integer echotype`
	`integer echosta`
	`integer echodat`

Inquire LID State	`integer function cfqlidstate(devclass, devnum, valid,`
	`1 state)`
	`integer devclass`
	`integer devnum`
	`integer valid`
	`integer state`

Table G-3 *SunCGI Fortran Binding – Part III— Continued*

CGI Specification Name	***Fortran Binding***

Inquire LID Capabilities	`integer function cfqlidcaps(devclass, devnum, valid,`
	`1 sample, change, numassoc, trigassoc, prompt,`
	`2 acknowledgement, echo, echotype, n, classdep, state)`
	`integer devclass`
	`integer devnum`
	`integer valid`
	`integer sample`
	`integer change`
	`integer numassoc`
	`integer trigassoc(*)`
	`integer prompt`
	`integer acknowledgement`
	`integer echo(*)`
	`integer echotype(*)`
	`integer n`
	`character*(*) classdep`
	`integer state(*)`

Inquire Line Attributes	`integer function cfqlnatts(style, width, color, index)`
	`integer style`
	`real width`
	`integer color, index`

Inquire Marker Attributes	`integer function cfqmkatts(type, size, color, index)`
	`integer type`
	`real size`
	`integer color, index`

Inquire Output Capabilities	`integer function cfqoutcap(first, last, list)`
	`integer first, last`
	`character*80 list(*)`

Inquire Output Function Set	`integer function cfqoutfunset(level, support)`
	`integer level`
	`integer support`

Inquire Pattern Attributes	`integer function cfqpatatts(cindex, row, column, colorlis,`
	`1 x, y, dx, dy)`
	`integer cindex`
	`integer row`
	`integer column`
	`integer colorlis(*)`
	`integer x`
	`integer y`
	`integer dx`
	`integer dy`

Table G-3 *SunCGI Fortran Binding – Part III— Continued*

CGI Specification Name	***Fortran Binding***

Inquire Physical Coordinate System	`integer function cfqphyscsys(name, xbase, ybase, xext, yext,`
	`1 xunits, yunits)`
	`integer name`
	`integer xbase, ybase`
	`integer xext, yext`
	`real xunits, yunits`

Inquire Pixel Array	`integer function cfqpixarr(px, py, m, n, colorind, name)`
	`integer px, py`
	`integer m, n`
	`integer colorind(*)`
	`integer name`

Inquire Text Attributes	`integer function cfqtextatts(fontset, index, cfont, prec,`
	`1 efac, space, color, hgt, bx, by, ux, uy, path, halign,`
	`2 valign, hfac, cfac)`
	`integer fontset, index, cfont, prec`
	`real efac, space`
	`integer color, hgt`
	`real bx, by, ux, uy`
	`integer path, halign, valign`
	`real hfac, cfac`

Inquire Text Extent	`integer function cfqtextext(string, nextchar,`
	`1 conx, cony, llpx, llpy, ulpx, ulpy, urpx, urpy)`
	`character*(*) string`
	`character*(*) nextchar`
	`integer conx`
	`integer cony`
	`integer llpx`
	`integer llpy`
	`integer ulpx`
	`integer ulpy`
	`integer urpx`
	`integer urpy`

Table G-3 *SunCGI Fortran Binding – Part III— Continued*

CGI Specification Name	***Fortran Binding***

Inquire Trigger Capabilities	`integer function cfqtrigcaps(trigger, valid, change, n,`
	`1 class, assoc, maxassoc, prompt, acknowledgement,`
	`2 name, description)`
	`integer trigger`
	`integer valid`
	`integer change`
	`integer n`
	`integer class(*)`
	`integer assoc(*)`
	`integer maxassoc`
	`integer prompt`
	`integer acknowledgement`
	`character*(*) name`
	`character*(*) description`

Inquire Trigger State	`integer function cfqtrigstate(trigger, valid, state, n,`
	`1 class, assoc)`
	`integer trigger`
	`integer valid`
	`integer state`
	`integer n`
	`integer class(*)`
	`integer assoc(*)`

Inquire VDC Type	`integer function cfqvdctype(type)` `integer type`

Interior Style	`integer function cfintstyle(istyle, perimvis)`
	`integer istyle`
	`integer perimvis`

Line Color	`integer function cflncolor(index)` `integer index`
Line Endstyle (SunCGI Extension)	`integer function cflnendstyle(ttyp)` `integer ttyp`
Line Type	`integer function cflntype(ttyp)` `integer ttyp`
Line Width Specification Mode	`integer function cflnspecmode(mode)` `integer mode`

Table G-4 *SunCGI Fortran Binding – Part IV*

CGI Specification Name	***Fortran Binding***
Line Width	`integer function cflnwidth(index)` `real index`
Marker Color	`integer function cfmkcolor(index)` `integer index`
Marker Size Specification Mode	`integer function cfmkspecmode(mode)` `integer mode`
Marker Size	`integer function cfmksize(index)` `real index`
Marker Type	`integer function cfmktype(ttyp)` `integer ttyp`
Open CGI (SunCGI Extension)	`integer function cfopencgi()`

Open View Surface (SunCGI Extension)	`integer function cfopenvws(name, screenname, windowname,`
	`1 windowfd, retained, dd, cmapsize, cmapname, flags,`
	`2 ptr)`
	`integer name`
	`character*(*) screenname`
	`character*(*) windowname`
	`integer windowfd`
	`integer retained`
	`integer dd`
	`integer cmapsize`
	`character*(*) cmapname`
	`integer flags`
	`character*(*) ptr`

Partial Polygon	`integer function cfppolygon(xcoors, ycoors, n, flag)`
	`integer xcoors(*)`
	`integer ycoors(*)`
	`integer n`
	`integer flag`

Pattern Index	`integer function cfpatix(index)` `integer index`
Pattern Reference Point	`integer function cfpatrefpt(x, y)` `integer x, y`
Pattern Size	`integer function cfpatsize(dx, dy)` `integer dx, dy`

Pattern Table	`integer function cfpattable(index, m, n, colorind)`
	`integer index`
	`integer m, n`
	`integer colorind(*)`

Table G-4 *SunCGI Fortran Binding – Part IV— Continued*

CGI Specification Name	***Fortran Binding***
Pattern with Fill Color (SunCGI Extension)	`integer function cfpatfillcolor(flag)` `integer flag`
Perimeter Color	`integer function cfperimcolor(index)` `integer index`
Perimeter Type	`integer function cfperimtype(ttyp)` `integer ttyp`
Perimeter Width Specification Mode	`integer function cfperimspecmode(mode)` `integer mode`
Perimeter Width	`integer function cfperimwidth(index)` `real index`

Pixel Array	`integer function cfpixarr(px, py, m, n, colorind)`
	`integer px, py`
	`integer m, n`
	`integer colorind(*)`

Polygon	`integer function cfpolygon(xcoors, ycoors, n)`
	`integer xcoors(*)`
	`integer ycoors(*)`
	`integer n`

Polyline Bundle Index	`integer function cfpolylnbundix(index)` `integer index`

Polyline	`integer function cfpolyline(xcoors, ycoors, n)`
	`integer xcoors(*)`
	`integer ycoors(*)`
	`integer n`

Polymarker Bundle Index	`integer function cfpolymkbundix(index)` `integer index`

Polymarker	`integer function cfpolymarker(xcoors, ycoors, n)`
	`integer xcoors(*)`
	`integer ycoors(*)`
	`integer n`

Rectangle	`integer function cfrectangle(xbot, ybot, xtop, ytop)` `integer xbot, ybot, xtop, ytop`

Release Input Device	`integer function cfrelidev(devclass, devnum)`
	`integer devclass`
	`integer devnum`

Table G-5 *SunCGI Fortran Binding – Part V*

CGI Specification Name	***Fortran Binding***

Request Input	`integer function cfreqinp(devclass, devnum, timeout,`
	`1 valid, x, y, xlist, ylist, n, val, choice, string,`
	`2 segid, pickid, trigger)`
	`integer devclass`
	`integer devnum`
	`integer timeout`
	`integer valid`
	`integer x, y`
	`integer xlist(*)`
	`integer ylist(*)`
	`integer n`
	`real val`
	`integer choice`
	`character*(*) string`
	`integer segid`
	`integer pickid`
	`integer trigger`

Reset to Defaults	`integer function cfrsttodefs()`

Sample Input	`integer function cfsampinp(devclass, devnum, valid, x, y,`
	`1 xlist, ylist, n, val, choice, string, segid, pickid)`
	`integer devclass`
	`integer devnum`
	`integer valid`
	`integer x, y`
	`integer xlist(*)`
	`integer ylist(*)`
	`integer n`
	`real val`
	`integer choice`
	`character*(*) string`
	`integer segid`
	`integer pickid`

Selective Flush of Event Queue	`integer function cfsflusheventqu(devclass, devnum)`
	`integer devclass`
	`integer devnum`

Set Aspect Source Flags	`integer function cfsaspsouflags(fval, fnum, n)` `integer fval(*), fnum(*), n`

Set Default Trigger Associations	`integer function cfsdefatrigassoc(devclass, devnum)`
	`integer devclass`
	`integer devnum`

Table G-5 *SunCGI Fortran Binding – Part V— Continued*

CGI Specification Name	***Fortran Binding***

Set Drawing Mode	`integer function cfsdrawmode(visibility, source,`
	`1 destination, combination)`
	`integer visibility`
	`integer source`
	`integer destination`
	`integer combination`

Set Error Warning Mask	`integer function cfserrwarnmk(action)` `integer action`
Set Global Drawing Mode (SunCGI Extension)	`integer function cfsgldrawmode(combination)` `integer combination`

Set Initial Value	`integer function cfsinitval(devclass, devnum, x, y,`
	`1 xlist, ylist, n, val, choice, string, segid, pickid)`
	`integer devclass`
	`integer devnum`
	`integer x, y`
	`integer xlist(*)`
	`integer ylist(*)`
	`integer n`
	`real val`
	`integer choice`
	`character*(*) string`
	`integer segid`
	`integer pickid`

Set Up SIGWINCH (SunCGI Extension)	`integer function cfsupsig(name, sig_function)`
	`integer name`
	`external sig_function`

Set VALUATOR Range	`integer function cfsvalrange(devnum, mn, mx)`
	`integer devnum`
	`real mn, mx`

Text Alignment	`integer function cftextalign(halign, valign, hcalind,`
	`1 vcalind)`
	`integer halign`
	`integer valign`
	`real hcalind, vcalind`

Text Bundle Index	`integer function cftextbundix(index)` `integer index`
Text Color	`integer function cftextcolor(index)` `integer index`
Text Font Index	`integer function cftextfontix(index)` `integer index`

Table G-5 *SunCGI Fortran Binding – Part V— Continued*

CGI Specification Name	***Fortran Binding***
Text Precision	`integer function cftextprec(ttyp)` `integer ttyp`

Text	`integer function cftext(x, y, string)`
	`integer x`
	`integer y`
	`character*(*) string`

Track Off	`integer function cftrackoff(devclass, devnum, tracktype,`
	`1 action)`
	`integer devclass`
	`integer devnum`
	`integer tracktype`
	`integer action`

Track On	`integer function cftrackon(devclass, devnum, echotype,`
	`1 exlow, eylow, exup, eyup, x, y, xlist, ylist, n, val,`
	`2 choice, string, segid, pickid)`
	`integer devclass`
	`integer devnum`
	`integer echotype`
	`integer exlow`
	`integer eylow`
	`integer exup`
	`integer eyup`
	`integer x, y`
	`integer xlist(*)`
	`integer ylist(*)`
	`integer n`
	`real val`
	`integer choice`
	`character*(*) string`
	`integer segid`
	`integer pickid`

VDC Extent	`integer function cfvdcext(xbot, ybot, xtop, ytop)` `integer xbot, ybot, xtop, ytop`

VDM Text	`integer function cfvdmtext(x, y, flag, string)`
	`integer x`
	`integer y`
	`integer flag`
	`character*(*) string`

H

Short C Binding

H

Short C Binding

At the time **SunCGI** was implemented, there was no official ANSI C binding for CGI. Sun Microsystems has tried to anticipate the eventual C binding with a set of shorter function names. The **SunCGI** binding is inspired by the C language binding of GKS. These names are contained in the header file `<cgicbind.h>` which must be included in an application program using the short C binding.

Table H-1 *Correspondence Between Long and Short C Names*

Long Name	*Short Name*
activate_vws	Cactvws
append_text	Captext
associate	Cassoc
await_event	Cawaitev
bitblt_pattern_array	Cbtblpatarr
bitblt_patterned_source_array	Cbtblpatsouarr
bitblt_source_array	Cbtblsouarr
cell_array	Ccellarr
character_expansion_factor	Ccharexpfac
character_height	Ccharheight
character_orientation	Ccharorientation
character_path	Ccharpath
character_set_index	Ccharsetix
character_spacing	Ccharspacing
circle	Ccircle
circular_arc_3pt	Ccircarcthree
circular_arc_3pt_close	Ccircarcthreecl
circular_arc_center	Ccircarccent
circular_arc_center_close	Ccircarccentcl
clear_control	Cclrcont
clear_view_surface	Cclrvws
clip_indicator	Cclipind
clip_rectangle	Ccliprect
close_cgi	Cclosecgi
close_vws	Cclosevws
color_table	Ccotable
deactivate_vws	Cdeactvws

Table H-1 *Correspondence Between Long and Short C Names— Continued*

Long Name	*Short Name*
define_bundle_index	Cdefbundix
device_viewport	Cdevvpt
disable_events	Cdaevents
disjoint_polyline	Cdpolyline
dissociate	Cdissoc
echo_off	Cechooff
echo_on	Cechoon
echo_update	Cechoupd
ellipse	Cellipse
elliptical_arc	Celliparc
elliptical_arc_close	Celliparccl
enable_events	Cenevents
fill_area_bundle_index	Cflareabundix
fill_color	Cflcolor
fixed_font	Cfixedfont
flush_event_queue	Cflusheventqu
get_last_requested_input	Cgetlastreqinp
hard_reset	Chardrst
hatch_index	Chatchix
initialize_lid	Cinitlid
initiate_request	Cinitreq
inquire_aspect_source_flags	Cqasfs
inquire_bitblt_alignments	Cqbtblalign
inquire_cell_array	Cqcellarr
inquire_device_bitmap	Cqdevbtmp
inquire_device_class	Cqdevclass
inquire_device_identification	Cqdevid
inquire_drawing_mode	Cqdrawmode
inquire_event_queue_state	Cqevquestate
inquire_fill_area_attributes	Cqflareaatts
inquire_input_capabilities	Cqinpcaps
inquire_lid_capabilities	Cqlidcaps
inquire_lid_state	Cqlidstate
inquire_lid_state_list	Cqlidstatelis
inquire_line_attributes	Cqlnatts
inquire_marker_attributes	Cqmkatts
inquire_output_capabilities	Cqoutcap
inquire_output_function_set	Cqoutfunset
inquire_pattern_attributes	Cqpatatts
inquire_physical_coordinate_system	Cqphyscsys
inquire_pixel_array	Cqpixarr
inquire_text_attributes	Cqtextatts
inquire_text_extent	Cqtextext
inquire_trigger_capabilities	Cqtrigcaps
inquire_trigger_state	Cqtrigstate
inquire_vdc_type	Cqvdctype

Table H-1 *Correspondence Between Long and Short C Names— Continued*

Long Name	*Short Name*
interior_style	Cintstyle
line_color	Clncolor
line_endstyle	Clnendstyle
line_type	Clntype
line_width	Clnwidth
line_width_specification_mode	Clnwidthspecmode
marker_color	Cmkcolor
marker_size	Cmksize
marker_size_specification_mode	Cmksizespecmode
marker_type	Cmktype
open_cgi	Copencgi
open_vws	Copenvws
partial_polygon	Cppolygon
pattern_index	Cpatix
pattern_reference_point	Cpatrefpt
pattern_size	Cpatsize
pattern_table	Cpattable
pattern_with_fill_color	Cpatfillcoior
perimeter_color	Cperimcolor
perimeter_type	Cperimtype
perimeter_width	Cperimwidth
perimeter_width_specification_mode	Cperimwidthspecmode
pixel_array	Cpixarr
polygon	Cpolygon
polyline	Cpolyline
polyline_bundle_index	Cpolylnbundix
polymarker	Cpolymarker
polymarker_bundle_Index	Cpolymkbundix
rectangle	Crectangle
release_input_device	Crelidev
request_input	Creqinp
reset_to_defaults	Crsttodefs
sample_input	Csampinp
selective_flush_of_event_queue	Cselectflusheventqu
set_aspect_source_flags	Csaspsouflags
set_default_trigger_associations	Csdefatrigassoc
set_drawing_mode	Csdrawmode
set_error_warning_mask	Cserrwarnmk
set_global_drawing_mode	Csgldrawmode
set_initial_value	Csinitval
set_up_sigwinch	Csupsig
set_valuator_range	Csvalrange
text	Ctext
text_alignment	Ctextalign
text_bundle_index	Ctextbundix
text_color	Ctextcolor

Table H-1 *Correspondence Between Long and Short C Names— Continued*

Long Name	*Short Name*
`text_font_index`	`Ctextfontix`
`text_precision`	`Ctextprec`
`track_off`	`Ctrackoff`
`track_on`	`Ctrackon`
`vdc_extent`	`Cvdcext`
`vdm_text`	`Cvdmtext`

Index

L

V

W

Revision History

Revision	*Date*	*Comments*
A	5/15/85	2.0 Production Release.
B	2/17/86	3.0 Production Release.
C	3/17/86	2.3 Production Release.