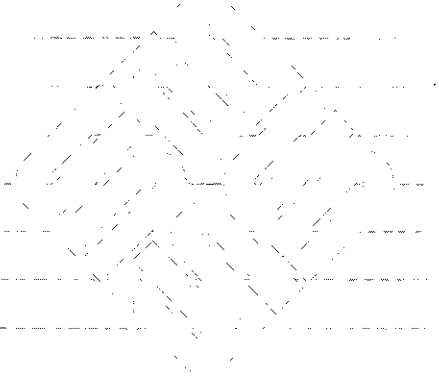
sun microsystems

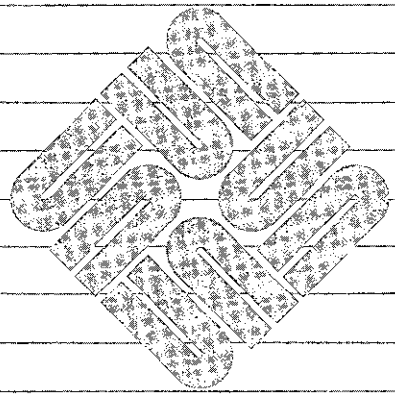
Programmer's Reference Manual *for* SunCGI



Sun Microsystems, Inc. • 2550 Garcia Avenue • Mountain View, CA 94043 • 415-960-1300

sun microsystems

Programmer's Reference Manual *for* SunCGI



Sun Microsystems, Inc. • 2550 Garcia Avenue • Mountain View, CA 94043 • 415-960-1300

Part No: 800-1166-01
Revision A of 15 May, 1985

Acknowledgements

Revision History

Rev	Date	Comments
A	15 May 1985	First release of this Programmer's Reference Manual.

Contents

Contents

Tables

Chapter 1

Introduction

SunCGI (Sun Computer Graphics Interface) provides access to low-level graphics device functions without the restrictions, benefits, or overhead of higher-level graphics packages such as **SunCore**. **SunCGI** is useful for two-dimensional 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 all of Sun's devices without explicitly specifying the device at compile time. Furthermore, **SunCGI** provides output primitives (for example, circles), attributes (for example, sophisticated pattern filling), and input primitives which are not offered by **SunCore**.

SunCGI is Sun Microsystem's interpretation of the March, 1984 working draft of the ANSI X3H3 committee which is commissioned with designing the CGI standard[1]. The CGI standard is currently under development, and therefore, CGI has not been accepted by the X3H3 committee, ANSI, or the computer graphics community. Furthermore, only certain models within the CGI proposed standard 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. Furthermore, the user should be aware that CGI does not support three-dimensional 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 such as **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. Notations Used in this Manual

Within the text of this manual, a `typewriter font` is used to represent function names and types. For example, `append_text` is the name of a function; `Cint` is the name of a type. The formal definitions of functions and arguments are printed in the `typewriter font` so that they resemble code as typed at a terminal.

---

[1] **Both the CGI standard and Sun's interpretation of it are subject to change.**

Italic font is used to indicate an internal state of **SunCGI**. Internal states of **SunCGI** are not explicitly enumerated, but their definition should be obvious from their names (for example, *current font*). Internal states are distinguishable from function arguments in that they do not use the underscore (_) character.

Italics are also used in the conventional manner (that is, to accentuate important words and phrases). **Boldface** is used for the names of Sun software packages such as **SunCGI** and **SunCore**.

In examples of what a user types at a terminal, a **`bold typewriter font`** is used to represent what the user types, and the `typewriter font` represents what the software displays in response.

1.2. Using SunCGI

To link **SunCGI** with your application program, put the following line in your Makefile:

```
cc test1.o -lcgi -lsunwindow -lpixrect -lm
```

where *test1.o* is the name of the application program. Similarly, if your application program uses other libraries, they must also be included in the Makefile line.

SunCGI uses a variety of structures and enumerated types. These types are defined in Appendix C. You should include the files <cgitypes.h>, and <cgiconstants.h> to provide the necessary type definitions and constants for your application program.

All **SunCGI** functions can be called by one of two names: the expanded name or the **C**-language binding name. The expanded name is the first name provided in the function description section, whereas the **C**-language binding name is specified in parenthesis. You must include the file <cgicbind.h> in your application program if you want to use the **C**-language binding name. The **C**-language binding names are an attempt to anticipate the **C**-language binding of CGI. However, no **C**-language binding has been included in the CGI standard, and the **SunCGI** binding is inspired by the **C**-language binding of GKS.

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.

1.3. Overview of SunCGI

This section provides an overview of the substance of the **SunCGI** 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) **Pixwins** and
2) FORTRAN.

1.3.1. Control (Initialization and Termination)

The chapter on control 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 of the control chapter 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 and deactivate commands (which turn a view surface on or off without closing the view surface). The functions discussed in the control chapter also define the range of normalized 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, the control chapter describes **SunCGI's** option for trapping SIGWINCH signals (generated by manipulating the window which the application program is using).

1.3.2. Output

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 in addition to those mentioned in the previous sentence. Geometrical output primitives are affected by attributes described in the following chapter (such as 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.

1.3.3. Attributes

Attributes 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; however, the options available for filling geometrical output primitives requires 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.

1.3.4. Input

SunCGI offers a standard interface for receiving input from the mouse and the keyboard. The CGI input model is based on the concept of logical input devices such as LOCATORS, VALUATORS,and STROKES. Logical input devices are bound to physical devices (such as mouse buttons) called triggers. 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 (wait for one event),

4) RESPOND_EVENT (report one event asynchronously), and

5) QUEUE_EVENT (put each event at the end of the *event queue*).

1.3.5. 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, begin by copying the example program in the example program appendix and develop your application program from this example. Further help can be obtained by referring to the tables at the end of the appendix specifying error messages. These tables list commonly encountered problems and how to solve them.

1.3.6. Appendices

The first five appendices are provided as an informational reference which may 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 **Pixwins** (the SunWindow System) and

2. between**SunCGI** and the FORTRAN programming language.

1.3.6.1. Reference Appendices

The first appendix explains the difference between **SunCGI** and **SunCore**. The next appendix 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. The third appendix provides the type definitions used by the **SunCGI** functions. The fourth appendix lists the error messages and possible strategies for eliminating them. This appendix also lists possible causes of simple run-time errors. Finally, the fifth appendix describes a sample program.

1.3.6.2. Description of Interfaces to SunCGI

The final two appendices describe the interfaces between **SunCGI** and other **Sun** software packages: **Pixwins** and FORTRAN. The first of the two interface appendices explains how to call **SunCGI** from application programs written on top of **Pixwins**. This interface allows **SunCGI** to write output primitives in different pixwins using different attributes. This interface is useful for application programs which wish to control different areas of the view surface independently. The final appendix 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 Technical Committee, 1984, 'Graphics Kernel System dpANS', ACM SIGGRAPH, February 1984, ACM, New York.

2) ANSI X3H3 Technical Committee, 1984, 'Virtual Device Interface dpANS', X3H3 84/45, March 1984, ANSI, New York.

Chapter 2

Initializing (and Terminating) SunCGI

2.1. View Surface Selection And Initialization

No functions for initializing and terminating devices are provided in the current CGI standard. Therefore, 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 by calling `open_vws`. **SunCGI** is capable of handling more than one view surface at once. However, output primitives are not displayed unless a view surface is ***activated.***

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[2] until `activate_vws` is called again (see the following example).

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

```
main ()
{
    Cvwsurf device1,device2;      /* declarations */
    Cint name1,name2;
    Ccoor bot,top,center;
    Cint radius;
    /* values of arguments are assumed to be set by you */

    open_cgi();                   /* start cgi */
    NORMAL_VWSURF(device1, PIXWINDD); /* black-and-white view surface */
    open_vws(&name1,&device1);    /* open device number 1 */
    NORMAL_VWSURF(device2, CG1DD); /* color view surface */
    open_vws(&name2,&device2);    /* open device number 2 */
    rectangle(&bot,&top);         /* draw a rectangle on both devices */
    deactivate_vws(name2);        /* deactivate  device number 2 */
    circle(&center,radius);       /* draw a circle on device no. 1 only */
    activate_vws(name2);          /* activate  device number 2 */
    circle(&center,2*radius);     /* draw a circle on both devices */
    close_vws(name1);             /* close device number 1 */
    close_vws(name2);             /* close device number 2 */
    close_cgi();                  /* close cgi */
}
```

2.1.1. open_cgi *(*Copencgi*)*

```
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.

ERRORS:

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

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.

Table 2-1: SunCGI Default States

State	*Value*
Range of VDC Space	(0–32767 in both X and Y directions)
Clip Indicator	ON
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)

Some of the entries discussed in Table 2-1 may be unfamiliar to you at this time. However, all of these concepts are explained in the course of this chapter. Further, each of these concepts are referenced in the index.

2.1.2. open_vws *(*Copenvws*)*

```
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. If you want to reinitialize the state of the view surface without reopening it, you should 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. You must set the view surface type by assigning the ***dd*** element of the ***devdd*** argument to the name of the appropriate device driver as in this example[3]:

```
Cvwsurf device;
    NORMAL_VWSURF(device, BW2DD); /* black-and-white view surface */
    open_vws(&name,&device);
```

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

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, if you want to open a window with some nonstandard parameters, or open a second window from a graphics tool you should read the following paragraphs. If you want to use an existing ***pixwin*** then you should skip the following paragraphs and read Appendix F instead.

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. You may also define other characteristics of the view surface 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 *CG1DD*). 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 black-and-white 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. Refer to the ***Programmers' Reference Manual for Sun Windows*** 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 CGI Tool) is created. The view surface descriptor returned in ***name*** refers to the CGI Tool. The ***ptr*** element specifies the size and placement of the CGI Tool. *Ptr* is a pointer to a nine element array of characters. 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:

5	ENOTOPOP	CGI not in proper state CGI shall be either in state CGOP, VSOP, or VSAC.
11	ENOWSTYP	Specified view surface type does not exist.
12	EVSISOPN	Specified view surface is open.

Table 2-2: Available View Surfaces

Surface Name	*Description*
BW1DD	for the Sun-1 monochrome display
BW2DD	for the Sun-2 monochrome display
CG1DD	for the Sun-1 color display
CG2DD	for the Sun-2 color display
PIXWINDD	for windows on the Sun-1 monochrome display
CGPIXWINDD	for windows on color display

Table 2-3: View Surface Default States

State	*Value*
View Surface	(Cleared)
Device Viewport	(Total Screen)

2.1.3. activate_vws *(Cactvws)*

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

activate_vws activates the view surface specified by name. Future **SunCGI** calls affect this view surface. ***Nothing is displayed on a view surface unless that view surface is active.*** Note that activating a view surface may reset the state of **SunCGI.**

ERRORS:

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

13	EVSNOTOP	Specified view surface not open.
14	EVSISACT	Specified view surface is active.

2.1.4. deactivate_vws *(Cdeactvws)*

```
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 at a later time without having to be reopened. Note that activating a view surface may reset the state of **SunCGI**.

ERRORS:

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

2.1.5. close_vws *(Cclosevws)*

```
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:

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

2.1.6. close_cgi *(Cclosecgi)*

```
Cerror close_cgi()
```

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

It is recommended that you include a call to `close_cgi` in the exit routines of your application program to guarantee leaving the window system and **SunCGI** in a stable state.

ERRORS:

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

2.2. 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-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 the chapter on attributes. 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.

2.2.1. `inquire_device_identification` *(*`Cqdevid`*)*

```
Cerror inquire_device_identification (name,devid)
    Cint name;     /* device name */
    Cchar *devid; /* 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 six Sun Workstation types: *BW1DD*, *BW2DD*, *CG1DD*, *CG2DD*, *PIXWINDD*, or *CGPIXWINDD*. 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:

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

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

2.2.2. `inquire_device_class` *(*`Cqdevclass`*)*

```
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 such as `inquire_output_capabilities`.

ERRORS:

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

2.2.3. `inquire_physical_coordinate_system` *(*`Cqphyscsys`*)*

```
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; /* number of pixels in each direction */
    Cfloat *xunits,*yunits; /* number of pixels per mm. */
```

`inquire_coordinate_system` reports the physical dimensions of the coordinate system of view surface ***name*** in pixels and millimeters. The `inquire_coordinate_system` function is provided to permit the drawing of objects of a known physical size. `inquire_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. If you want to compute the actual size of the screen, you may obtain the number of pixels in the ***x*** and ***y*** directions from the monitor specifications and perform the division in your application program.

ERRORS:

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

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

2.2.4. inquire_output_function_set *(*Cqoutfunset*)*

```
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
} Vsuptype;
```

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:

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

2.2.5. inquire_vdc_type *(*Cqvdctype*)*

```
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). Use **SunCore** if you want to use a system with coordinate space expressed in real numbers.

ERRORS:

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

2.2.6. `inquire_output_capabilities` *(*Cqoutcap*)*

```
Cerror inquire_output_capabilities (first,num,list)
    Cint first,num; /* first and number elements
                       of 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:

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

2.2.7. 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.

2.2.7.1. `inquire_input_capabilities` *(*Cqinpcaps*)*

```
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 by the specific view surface. 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:

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

2.2.7.2. inquire_lid_capabilities *(*Cqlidcaps*)*

```
Cerror inquire_lid_capabilities (devclass,devnum,valid,table)
    Cdevoff devclass;
    Cint devnum; /* device type, 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;
    Clogical prompt;
    Clogical acknowledgement;
    Cechoav *echo;
    Cchar *classdep;
    Cstatelist state;
} Cliddescript;
```

The elements of ***table*** which are of type ***Clogical*** indicate whether an ability is present in a specified logical input device. The ***change*** element reports whether associations are changeable at all (all input devices except string are not changeable). The ***numassoc*** and ***trigassoc*** elements of ***table*** report how many and which triggers are associated with the specified logical input device. The ***echo*** argument describes which echo types are supported (see the chapter on input 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 2-4: 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:

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

2.2.7.3. `inquire_trigger_capabilities` *(*`Cqtrigcaps`*)*

```
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.

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

```
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*** and ***trigassoc*** elements of ***tdis*** report how many and which logical input devices can be associated with ***trigger***. The ***maxassoc*** element reports the maximum number of devices that can be associated with a particular trigger. The ***prompt*** and ***acknowledgement*** elements report the ability of the trigger to support these abilities. **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:

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

2.3. 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**.

2.3.1. Coordinate Definition

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.

2.3.1.1. vdc_extent *(Cvdcext)*

```
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 you, 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 you to translate (move) or scale (zoom in/zoom out) the output primitives. However, no rotation functions are provided by **SunCGI**, and therefore, must be built by you. The code fragment below translates and zooms in on a rectangle:

```
#include <cgidefs.h>
main()
{
Cvwsurf device;
Cint name;
Ccoor upper,lower;
Cint xl,yl,xu,yu;
Ccoor dv1,dv2;

        device.dd = PIXWINDD;   /* turn on a pixwin */
        open_cgi();
        open_vws(&name,&device);
        activate_view_surface(name);
        lower.x=30; /* coordinates of rectangle */
        lower.y=30;
        upper.x=70;
        upper.y=70;
        dv1.x = 0;
        dv1.y = 0;
        dv2.x = 200;
        dv2.y = 200;
        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(4);
        close_view_surface(name);
        close_cgi();
}
```

ERRORS:

5	ENOTOPOP	CGI not in proper state CGI shall be either in state CGOP, VSOP, or VSAC.
20	EBADRCTD	Rectangle definition is invalid.
24	EVDCSDIL	VDC space definition is illegal.

2.3.1.2. `device_viewport` *(*`Cdevvpt`*)*

```
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:

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

2.3.2. Clipping

For some application programs, it is desirable to have clipping 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 clipping area is set by the `clip_rectangle` function.

2.3.2.1. `clip_indicator` *(*`Cclipind`*)*

```
Cerror clip_indicator(cflag)
    Cclip cflag; /* CLIP_RECTANGLE, VDC_EXTENT, or OFF */
```

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

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

ERRORS:

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

2.3.2.2. `clip_rectangle` *(*`Ccliprect`*)*

```
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 clipping rectangle impacts all view surfaces.

ERRORS:

5	ENOTOPOP	CGI not in proper state CGI shall be either in state CGOP, VSOP, or VSAC.
20	EBADRCTD	Rectangle definition is invalid.
22	ECLIPTOL	Clipping rectangle is too large.
23	ECLIPTOS	Clipping rectangle is too small.

2.3.3. Device Control

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.

2.3.3.1. `hard_reset` *(*`Chardrst`*)*

```
Cerror hard_reset()
```

`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:

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

2.3.3.2. `reset_to_defaults` *(*`Crsttodefs`*)*

```
Cerror reset_to_defaults()
```

`reset_to_defaults` returns output attributes to defaults. `reset_to_defaults` does ***not*** clear the screen, reset the input devices, or reset the ***character set index.***

ERRORS:

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

2.3.3.3. `clear_view_surface` *(*`Cclrvws`*)*

```
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:

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

2.3.3.4. `clear_control` *(Cclrcont)*

```
Cerror clear_control(soft,hard,intern,extent)
    Cacttype soft,hard; /* soft-copy action, hard-copy action */
    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 `clear_extent` is set to either VIEW_SURFACE, VIEWPORT, or CLIP_RECTANGLE and determines what area of the screen is cleared. The default states are CLEAR (***soft***), NO_OP (***hard***), RETAIN (***intern***), and VIEW_SURFACE (***extent***).

ERRORS:

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

2.3.3.5. `set_error_warning_mask` *(Cserrwarnmk)*

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

`set_error_warning_mask`[7] 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 user is advised ***not*** to set the ***action*** argument to NO_ACTION.

If the ***action*** argument is set to POLL, errors are detected and the returned value of the function is set to the error number. Error handling is therefore, left up to the application programmer. If the ***action*** is set to INTERRUPT an error message is printed on the terminal, and the program is stopped if the error is FATAL (See Appendix D). The default `error_warning_mask` is INTERRUPT.

ERRORS:

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

[7] 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.

2.4. Running SunCGI in the Window System

Two aspects of **SunCGI** pertaining to the window system are:

1. varying the size of graphics windows, and
2. handling SIGWINCHes (interrupt signals generated by changes to the view surface made by the window system).

2.4.1. Changing the Window Size

When the window size changes during execution time, the scale factors which map VDC Space into screen space are modified. More importantly, unless the application program has explicitly opened a view surface as a retained view surface, overlapping other windows destroys the picture in the graphics subwindow below, In this case, the picture must be regenerated by re-running the application program.

2.4.2. Passing SIGWINCHes to the Application Program

When a view surface is initialized or changed, the SIGWINCH signal is normally trapped by **SunCGI**. However, the SIGWINCH signal is passed up to the application program if the application program provides a SIGWINCH handling routine by using the `set_up_sigwinch` function. Under no circumstances will the user be able to access the SIGWINCHes generated when the view surface is initialized. Therefore, the `set_up_sigwinch` routine is primarily intended for users who want to activate a display list for picture regeneration, for example, when the size of the view surface has changed. If the view surface is retained, the pictures is redisplayed if the size of the view surface has not changed.

2.4.3. `set_up_sigwinch` *(*`Csupsig`*)*

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

`set_up_sigwinch` is a nonstandard **SunCGI** function that allows the application program to trap SIGWINCHes for viewsurface `name`. If the `sig_function` argument is nonzero, all SIGWINCHes which are not trapped by the internals of **SunCGI** (that is, view surface initialization, and changing the size of the window) are passed to the function specified by `sig_function`.

ERRORS:

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

Chapter 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[8].

Raster Primitives
draw text and scaled and unscaled two-dimensional arrays. The coordinate system for raster primitives depends on the type of primitive: cell array, pixel array, or bitblt (bit block transfer). 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 two-dimensional in keeping with the CGI standard. However, polygons with holes (via the `partial_polygon` function) are provided in order to support three-dimensional graphics packages.

3.1.1. Polygonal Primitives

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

[8] 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 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 the next chapter.

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. However, the ***rectangle*** function implicitly generates closed objects[9].

3.1.1.1. `polyline` *(*`Cpolyline`*)*

```
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 (255), an error is generated.

ERRORS:

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

3.1.1.2. `disjoint_polyline` *(*`Cdpolyline`*)*

[9] The only instance in which ***rectangle*** does not produce a closed object is if one of the corners exceeds a clipping boundary.

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

`disjoint_polyline` draws lines between pairs of elements in ***ptlist***. 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 ***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 256, an error is generated. `disjoint_polyline` is typically used to implement scan-line polygon filling algorithms.

ERRORS:

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

3.1.1.3. `polymarker` *(*`Cpolymarker`*)*

```
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:

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

3.1.1.4. `polygon` *(*`Cpolygon`*)*

```
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 pattern and color used to fill the polygon are set by the functions described in the solid attributes section of the attributes chapter. The characteristics of the edges are controlled by the perimeter attribute functions and ***not*** the line attribute functions (which regulate the drawing of ***polylines***). The number of points in the polygon used to determine the error condition of too few points is the total number of points on the ***global polygon list*** not the number of points specified in ***polycoors***.

ERRORS:

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

3.1.1.5. `partial_polygon` *(*Cppolygon*)*

```
Cerror partial_polygon(polycoors, flag)
    Ccoorlist *polycoors; /* list of points */
    Ccflag flag; /* add to point buffer */
```

`partial_polygon` adds elements to the ***global polygon list without*** displaying the polygon. The flag controls whether the ***previous*** polygon on the ***global polygon list*** is open or closed. If the *cflag* is set to CLOSED, the last polygon on the ***global polygon list*** is closed by drawing a ***visible*** perimeter edge between the last and the first points of the last polygon in the ***global polygon list.*** If the *cflag* is set to OPEN, an ***invisible*** perimeter edge is drawn between the last and the first points of the last polygon in the ***global polygon list.*** The `partial_polygon` function provides the capability of drawing multiple-boundary polygons, including polygons with holes.

```
#include "cgidefs.h"
{
        .
        .
        .
    Ccoor list[4];
    Ccoorlist points;
        .
        .
        .
    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); /* draw large solid square */
    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 */
}
```

An error is detected if the number of points on the ***global polygon list*** exceeds 255. 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:

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

9.1.1.6. `rectangle` *(*`Crectangle`*)*

```
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***. The fill pattern is determined by the solid object attribute functions. Calls to ***rectangle*** do not affect the ***global polygon list***. The interior of the rectangle (the filled portion) is defined by ***lrc*** and ***ulc***. The perimeter is drawn outside of this region. The appearance of the rectangle is determined by the fill area and perimeter attributes.

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 ***lrc*** and ***ulc*** are reversed, the points are automatically reversed and the rectangle is drawn normally.

ERRORS:

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

3.1.2. Conical Primitives

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.

3.1.2.1. `circle` *(*`Ccircle`*)*

```
Cerror circle(c1, rad)
    Ccoor *c1; /* center */
    Cint rad; /* radius */
```

circle draws a circle of radius ***rad*** centered at ***c1***. 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 ***inside*** 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:

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

3.1.2.2. `circular_arc_center` *(*`Ccircarccent`*)*

```
Cerror circular_arc_center(c1, c2x, c2y, c3x, c3y, rad)
    Ccoor *c1; /* center */
    Cint c2x,c2y,c3x,c3y; /* endpoints */
    Cint rad; /* radius */
```

`circular_arc_center` draws a circular arc through 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. If ***c2x, c2y*** or ***c3x, c3y*** are not on the circumference of the circle, an error is generated and the arc is not drawn. Circular 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, c2x, c2y,*** and ***c3x, c3y*** and the acute angle specified by these same points. Therefore switching the values of ***c2x, c2y*** and ***c3x, c3y*** will produce complementary arcs.

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 perimeter 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 poly***lines***.

ERRORS:

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

3.1.2.3. `circular_arc_center_close` *(*`Ccircarccentcl`*)*

```
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 ***c3x, 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 (that is, it uses the ***fill area attributes***). The CHORD algorithm draws a line between the endpoints of the arc and then fills this region. `circular_arc_center_close` is useful for drawing pie charts (see following example):

```
#include <cgidefs.h>

    { /* draws four quadrants in different colors */
    Cint radius;
    Ccoor c1;

    interior_style(SOLIDI,OFF) /* set arc type to SOLID and no visible perimeter */
    c1.x = 16000; /* center */
    c1.y = 16000;
    radius =8000; /* radius */
    fill_color(1); /* color of quadrant 1 */
    circular_arc_center_close(&c1,16000,24000,24000,16000,radius,PIE);
    fill_color(2); /* color of quadrant 2 */
    circular_arc_center_close(&c1,16000,24000,12000,16000,radius,PIE);
    fill_color(3); /* color of quadrant 3 */
    circular_arc_center_close(&c1,12000,16000,12000,16000,radius,PIE);
    fill_color(4); /* color of quadrant 4 */
    circular_arc_center_close(&c1,12000,16000,24000,16000,radius,PIE);

    sleep(10);

    }
```

ERRORS:

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

3.1.2.4. `circular_arc_3pt` *(*`Ccircarcthree`*)*

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

`circular_arc_3pt` draws a circular arc starting at point *c1* and ending at point *c3* which is ***guaranteed*** to pass through point *c2*. 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:

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

3.1.2.5. `circular_arc_3pt_close` *(Ccircarcthreecl)*

```
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***. 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:

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

3.1.2.6. `ellipse` *(Cellipse)*

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

ellipse draws an ellipse centered at point ***c1*** with major(x) and minor (y) axes which terminate at ***majx*** and ***miny***[10]. 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 same as those attributes which control the drawing of circles and rectangles (namely, the perimeter and fill area attributes).

ERRORS:

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

3.1.2.7. `elliptical_arc` *(Celliparc)*

```
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 which terminate at ***majx*** and ***miny***. ***Sx, sy*** and ***ex, ey*** are the starting and 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

[10] 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.

perimeter of the ellipse. Circular 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.

ERRORS:

4	ENOTVSAC	CGI not in proper state: CGI shall be in state VSAC.
65	EARCPNEL	Arc points do not lie on ellipse.

3.1.2.8. `elliptical_arc_close` *(*`Celliparccl`*)*

```
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; /* enpoints 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_close` 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:

4	ENOTVSAC	CGI not in proper state: CGI shall be in state VSAC.
65	EARCPNEL	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.

3.2.1. `text` *(*`Ctext`*)*

```
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 the next chapter; however, it is worth noting some of the effects of the most important attribute, ***text precision***. The size of the text depends on the setting of the ***text precision*** (see the `text_precision` function) and the font selected. The results of clipping also depend on the ***text precision*** attribute (unless the ***text precision*** is set to STROKE). Control characters are displayed as blanks, except in the SYMBOL font where they may be drawn as pictures of bugs.

ERRORS:

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

3.2.2. `vdm_text` *(Cvdmtext)*

```
Cerror vdm_text(c1, flag, tstring)
    Ccoor c1; /* 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 ***c1*** (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:

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

3.2.3. `append_text` *(Captext)*

```
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:

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

3.2.4. inquire_text_extent *(Cqtextext)*

```
Cerror inquire_text_extent(tstring, nextchar, concat, lleft, uleft, uright)
    Cchar *tstring; /* text */
    Cchar nextchar; /* last character */
    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***[11]. If ***nextchar*** equals "", the last point of the current character is used. The argument ***concat*** returns the coordinates of the point where appended text would start. The arguments ***lleft, uleft***, and ***uright*** return the coordinates of the bounding box of text contained in ***tstring***.

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:

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

3.2.5. cell_array *(Ccellarr)*

```
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 three-by-three cell array is supposed to be 3.333 pixels wide, the actual cell array will be nine pixels wide instead of ten.

ERRORS:

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

4	ENOTVSAC	CGI not in proper state: CGI shall be in state VSAC.
66	ECELLATS	Cell array dimensions dx,dy are too small.
67	ECELLPOS	Cell array dimensions must be positive.

3.2.6. pixel_array *(*Cpixarr*)*

```
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:

4	ENOTVSAC	CGI not in proper state: CGI shall be in state VSAC.
69	EVALOVWS	Value outside of view surface.

3.2.7. bitblt_source_array *(*Cbtblsouarr*)*

```
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 pixel array (in VDC Space) */
    Cint xe,ye; /* dimensions of source pixel 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 ***pixrects***[12] which contain the bitmaps at the source and target destinations. 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:

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

[12] Refer to the *Programmers' Reference Manual for Sun Windows* for more information about pixrects.

69 EVALOVWS Value outside of view surface.

3.2.8. bitblt_pattern_array *(*Cbtblpatarr*)*

```
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 array ***pixtarget*** which is determined by ***ox, oy,*** and ***dx, dy.*** The pattern reference point determines the offset of pattern array from the point zero. The resultant pattern array is displayed at ***ox, oy.*** If the replicated pattern array overlaps with the source array on the screen, the visual result depends on the current drawing mode.

Pixpat is a pointer to a pixrect which must be created by the application program. If ***pixpat*** is not a pointer to an existing pixrect, an error is generated. ***Piztarget*** is a pointer to a pixrect which is created by a bitblt_pattern_array.

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

ERRORS:

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

3.2.9. bitblt_patterned_source_array *(*Cbtblpatsouarr*)*

```
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 the pattern (using the current ***drawing mode***) stored in ***pixpat*** to fill the area of array ***piztarget*** which is determined by ***ox, oy,*** and ***dx, dy.*** The replicated pattern array is ANDed into the pixrect pointed to by ***pixsource*** before the array is copied to ***pixtarget.*** The pattern reference point determines the offset of pattern array from the

point zero. The resultant pattern array is displayed at ***ox, oy***. If the replicated pattern array overlaps with the source array on the screen, the visual result depends on the current drawing mode. ***Pixpat*** is a pointer to a pixrect which must be created by your application program. ***Pixsource*** and ***pixtarget*** are pointers to pixrects which are created by a `bitblt_patterned_source_array`.

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

ERRORS:

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

3.2.10. `inquire_cell_array` *(*`Cqcellarr`*)*

```
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:

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

3.2.11. inquire_pixel_array *(Cqpixarr)*

```
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:

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

3.2.12. inquire_device_bitmap *(Cqdevbtmp)*

```
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. If you want to use subareas of this pixrect or manipulate it any other way, refer to the Pixrects Chapter in the ***Programmers' Reference Manual for Sun Windows***. Pixrects may be created in other manners; again refer to the Pixrect Chapter in the ***Programmers' Reference Manual for Sun Windows***.

ERRORS:

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

3.2.13. inquire_bitblt_alignments *(Cqbtblalign)*

```
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:

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

3.3.1. set_drawing_mode *(Csdrawmode)*

```
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 NOTted) replaces the destination pixrect. If ***combination*** is equal to AND, OR, NOT, or XOR the source pixrect and the destination pixrect are combined in the indicated Boolean fashion.

ERRORS:

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

3.3.2. set_global_drawing_mode *(Csgldrawmode)*

```
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 bitblt's*** 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.

ERRORS:

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

3.3.3. inquire_drawing_mode *(Cqdrawmode)*

```
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:

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

Chapter 4

Attributes

The current attributes determine how output primitives are displayed. Attributes are ***not*** specific to any view surface, but affect all view surfaces. If you want to avoid using the attribute functions, the current attributes are set to their default attributes which 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***). 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 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 ASFs	INDIVIDUAL	All Bundle Indices	1
Line Type	SOLID	Line Endstyle	POINT
Line Width Specification Mode	SCALED	Line Width	0.0
Line Color	1		
Marker Size Specification Mode	SCALED	Marker Size	0.0
Marker Color	1		
Fill Style	HOLLOW	Fill Visible	ON
Fill Color	1	Fill Hatch Index	0
Fill Pattern Index	1	Number of Pattern Table Entries	2
Pattern Reference Point	0,0	Pattern Size	300,300
Perimeter Style	SOLID	Perimeter Width Specification Mode	SCALED
Perimeter Width	0.0	Perimeter Color	1
Fontset	1	Current Font	STICK
Text Precision	STRING	Character Expansion Factor	1.0
Character Space	0.1	Character Color	1
Character Height	1000	Character Base.*x*	0.0
Character Base.*y*	0.0	Character Up.*x*	1.0
Character Up.*y*	1.0	Character Path	RIGHT
Horizontal Text Alignment	NRMAL	Vertical Text Alignment	NORMAL
Text Continuous Alignment.*x*	1.0	Text Continuous Alignment.*y*	1.0

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***.

4.1.1. `set_aspect_source_flags` *(*`Csaspsouflags`*)*

```
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*. 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:

5 ENOTOPOP 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

4.1.2. `define_bundle_index` *(*`Cdefbundix`*)*

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

`define_bundle_index` is a nonstandard function which 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. You should keep this fact in mind if you are designing display list traversal algorithms using **SunCGI**.

```
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;
```

ERRORS:

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

4.2. Line Attributes

SunCGI provides for specifying the style, width and color of lines which constitute poly***lines***, rectangles, circular arcs, and elliptical arcs. The functions do ***not*** affect the drawing of the perimeter of poly***gons***, rectangles, circles, or ellipses. The attributes of the perimeters of solid objects are set by the perimeter functions.

4.2.1. polyline_bundle_index *(*Cpolylnbundix*)*

```
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` function 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:

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

33 EBADLINX Polyline index is invalid.

4.2.2. line_type *(*Clntype*)*

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

The styles of line offered are SOLID, DASHED, DOTTED, DASHED_DOTTED, and DASH_DOT_DOTTED. The default line style is SOLID. The actual representation of a line on the screen is affected by the ***line endstyle.***

ERRORS:

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

30 EBTBUNDL ASF is BUNDLED.

4.2.3. line_endstyle *(*Clnendstyle*)*

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

line_endstyle determines how a textured (non-SOLID) line terminates. The three values which ***ttyp*** can assume are NATURAL, POINT, and BEST_FIT. 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:

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

4.2.4. line_width_specification_mode *(*Clnwidthspecmode*)*

```
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 (which can either be ABSOLUTE or SCALED). If the ***line width specification mode*** is changed from ABSOLUTE to SCALED, the change in the line width will probably be dramatic.

If multiple view surfaces are open, the line width is calculated on the basis of the first view surface opened.

ERRORS:

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

4.2.5. line_width *(*Clnwidth*)*

```
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:

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

4.2.6. line_color *(*Clncolor*)*

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

line_color resets the color of the lines. ***index*** selects an entry in the color lookup table. The default value of ***index*** is 1.

ERRORS:

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

4.3.1. polymarker_bundle_index *(*Cpolymkbundix*)*

```
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 width*** 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:

5	ENOTOPOP	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.
37	EBADMRKX	Polymarker index is invalid.

4.3.2. marker_type *(*Cmktype*)*

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

marker_type sets the marker type to one of five marker types DOT, PLUS, ASTERISK, CIRCLE, or X. Be sure to use an argument of Cmartype and not Cint.

ERRORS:

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

4.3.3. marker_size_specification_mode *(*Cmksizespecmode*)*

```
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 (which can either be ABSOLUTE or SCALED).

If multiple view surfaces are open, the marker size is calculated on the basis of the first view surface opened.

ERRORS:

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

4.3.4. marker_size *(Cmksize)*

```
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 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:

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

4.3.5. marker_color *(Cmkcolor)*

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

`marker_color` resets 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.

ERRORS:

5	ENOTOPOP	CGI not in proper state CGI shall be in state VDOP, VSOP, or VSAC.
30	EBTBUNDL	ASF is BUNDLED.
35	ECINDXLZ	Color index is less than zero.
36	EBADCOLX	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
: determine the general method for 'coloring-in' solid geometrical objects.

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

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

4.4.1. Fill Area Attributes

The fill area attribute functions determine the general method for drawing polygons, filled rectangles, circles, and ellipses.

4.4.1.1. `fill_area_bundle_index` *(*`Cflareabundix`*)*

```
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:

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

4.4.1.2. `interior_style` *(*`Cintstyle`*)*

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

`interior_style` sets the ***fill style*** for solid objects to either HOLLOW, SOLIDI, PATTERN, or HATCH. If the ***fill style*** is set to SOLIDI, the solid object is filled with the current ***fill color***. However, the appearance of the solid object on the screen depends on the ***drawing mode***. 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:

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

4.4.1.3. `fill_color` *(*`Cflcolor`*)*

```
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.

ERRORS:

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

4.4.2. Pattern Attributes

Geometrical primitives can be filled with two dimensional 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 two-dimensional array of color values and the X and Y dimensions of the array. The starting position (lower 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.

4.4.2.1. `hatch_index` *(*`Chatchix`*)*

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

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

ERRORS:

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

4.4.2.2. pattern_index (Cpatix)

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

`pattern_index` determines which index in the pattern table is used to fill solid objects when the ***fill style*** is set to PATTERN. `pattern_index` also determines the pattern which is used by ***drawing mode***. The default ***pattern index*** is 2. An error is generated if ***index*** points to an undefined entry in the pattern table. Note that `pattern_index` 0 is not defined.

ERRORS:

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

4.4.2.3. pattern_table (Cpattable)

```
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 10). 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. All nonzero entries in ***colorind*** are set to 1. Pattern 1 is initially defined to be a three-by-three matrix which is set to zero at the corners and one elsewhere. Pattern 1 produces simple cross-hatching. Pattern 2 (which produces a polka-dot pattern) is initially defined to be a three-by-three matrix which is set to 1 at the center and 0 elsewhere. The maximum number of elements in a pattern is 256.

ERRORS:

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

4.4.2.4. pattern_reference_point (Cpatrefpt)

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

`pattern_reference_point` defines the point in VDC Space where the ***pattern box*** begins. ***begin*** determines the offset of the pattern. ***pattern_reference_point*** has no effect if the ***interior style*** is set to HATCH. The lower left-hand corner of the ***pattern box*** is determined by ***begin***. The pattern is conceptually replicated over all of VDC Space — the ***begin*** point provides for fine positioning of the pattern. The default ***pattern reference point*** is (0,0).

ERRORS:

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

4.4.2.5. `pattern_size` *(*`Cpatsize`*)*

```
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, an error is generated and the pattern size is set ***m*** pixels by ***n*** pixels.[13] 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:

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

4.4.3. Perimeter Attributes

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 perform the same tasks. Perimeter attribute functions have no effect if the ***perimeter visibility*** is set to OFF. The perimeter attributes are essentially the same as the line attributes except that they affect the borders of solid attributes. Which attributes affect which primitives may become ambiguous when the ***interior style*** is set to HOLLOW.

4.4.3.1. `perimeter_type` *(*`Cperimtype`*)*

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

The styles of perimeter offered are SOLID, DASHED, DOTTED, DASHED_DOTTED, and DASH_DOT_DOTTED. The default perimeter style is SOLID. The actual representation of a

[13] ***m*** and ***n*** are obtained from the ***pattern table***.

perimeter is effected by ***drawing mode*** as well as the ***perimeter style.*** Notice that there is no ending style for perimeter. The endstyle is controlled by the ***line endstyle*** function.

ERRORS:

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

4.4.3.2. `perimeter_width` *(*`Cperimwidth`*)*

```
Cerror perimeter_width(index)
    Cfloat index; /* 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:

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

4.4.3.3. `perimeter_width_specification_mode` *(*`Cperimwidthspecmode`*)*

```
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.

If multiple view surfaces are open, the perimeter width is calculated on the basis of the first view surface opened.

ERRORS:

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

4.4.3.4. `perimeter_color` *(*`Cperimcolor`*)*

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

`perimeter_color` resets 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:

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

4.5. 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.

4.5.1. `text_bundle_index` *(*`Ctextbundix`*)*

```
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:

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

4.5.2. `text_precision` *(*`Ctextprec`*)*

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

`text_precision` controls the type of text displayed. ***ttyp*** can assume one of three values: STRING, CHARACTER, or STROKE. If the ***text precision*** is set to STRING, the firmware character set is used.

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. All text attributes have a visible effect on software generated characters, but according to the standard, text drawn with CHARACTER precision are not clipped in parts. However, in **SunCGI**, characters are clipped in parts. 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:

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

4.5.3. character_set_index *(*Ccharsetix*)*

```
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:

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

4.5.4. text_font_index *(*Ctextfontix*)*

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

text_font_index resets 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:

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

Table 4-3: Available Fonts

Font	*Available in String Precision?*
ROMAN	YES
GREEK	YES (displayed as STICK font)
SCRIPT	YES
OLDENGLISH	NO
STICK	YES
SYMBOLS	NO

4.5.5. `character_expansion_factor` *(*`Ccharexpfac`*)*

```
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 .01 or greater than 10.

ERRORS:

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

4.5.6. `character_spacing` *(*`Ccharspacing`*)*

```
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 .01 or greater than 10.

ERRORS:

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

4.5.7. character_height *(Ccharheight)*

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

The ***character_height*** function determines the height of text in VDC coordinates. 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:

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

4.5.8. fixed_font *(Cfixedfont)*

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

fixed_font is a ***nonstandard*** CGI function which allows characters to be of fixed or variable size. If ***index*** 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.

ERRORS:

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

4.5.9. text_color *(Ctextcolor)*

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

text_color resets 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:

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

36 EBADCOLX Color index is invalid.

4.5.10. character_orientation *(*Ccharorientation*)*

```
Cerror character_orientation (xup,yup,xbase,ybase)
    Cfloat xup,yup,xbase,ybase; /* character base and character 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 ***xbase*** and ***xup***. The bottom of the character box lies on an invisible line called the ***character base vector*** whose slope is determined by ***ybase*** and ***yup***.

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 *xup*=1.0, *yup*=1.0, *xbase*=0.0, and *ybase*=0.0.

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:

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

50 ECHRUPVZ Length of character up vector or character base vector is zero.

4.5.11. character_path *(*Ccharpath*)*

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

The ***character_path*** function respecifies direction that text is written. The four possible directions are RIGHT, LEFT, UP, and DOWN. 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:

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

4.5.12. text_alignment *(Ctextalign)*

```
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. The value of the ***halign*** argument (which must be one of LFT, CNTER, RGHT, NRMAL, CNT) 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***). 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 CTR the horizontal position of the text will begin equidistant from the right and the left edges of the text box. NRML 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*** percent of the width of the text box from the left edge of the character box. The default value of ***halign*** is NRML.

The value of the ***valign*** argument (which must be one of TOP, CAP, HALF, BASE, BOTTOM, NORMAL, CONT) specifies where the character is placed in relation to the *y* component of the text position. 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.[14] 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.[15] 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*** percent of the distance from the bottom edge of the character box. The default value of ***valign*** is NORMAL.

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

[15] 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:

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

4.6. Color Attributes

SunCGI supports only one color specification mode — INDEXED. Index 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 black and white, 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

4.6.1. `color_table` *(*`Ccotable`*)*

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

`color_table` resets ***color lookup table*** entries. The color lookup table is a set of 256 ***rgb*** entries. 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 modified. An error is generated if either the indices to the ***color lookup table*** are out of range.

ERRORS:

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

4.7. 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).

4.7.1. inquire_line_attributes *(*Cqlnatts*)*

```
Clinatt *inquire_line_attributes();
    /* 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;
```

Since **inquire_line_attributes** does not return an error, errors are only reported in INTERRUPT mode.

ERRORS:

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

4.7.2. inquire_marker_attributes *(*Cqmkatts*)*

```
Cmarkatt *inquire_marker_attributes();
    /* 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;
```

Since `inquire_marker_attributes` does not return an error, errors are only reported in INTERRUPT mode.

ERRORS:

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

4.7.3. `inquire_fill_area_attributes` *(*`Cqflareaatts`*)*

```
Cfillatt *inquire_fill_area_attributes();
    /* pointer to fill area attribute structure */
```

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;
```

Since `inquire_fill_area_attributes` does not return an error, errors are only reported in INTERRUPT mode.

ERRORS:

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

4.7.4. `inquire_pattern_attributes` *(*`Cqpatatts`*)*

```
Cpatternatt *inquire_pattern_attributes();
    /* 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;
```

Since `inquire_pattern_attributes` does not return an error, errors are only reported in INTERRUPT mode.

ERRORS:

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

4.7.5. `inquire_text_attributes` *(*`Cqtextatts`*)*

```
Ctextatt *inquire_text_attributes();
    /* 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;
```

Since `inquire_text_attributes` does not return an error, errors are only reported in INTERRUPT mode.

ERRORS:

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

4.7.6. inquire_aspect_source_flags (Cqasfs)

```
Cflaglist *inquire_aspect_source_flags()
    /* 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;
```

Since `inquire_aspect_source_flags` does not return an error, errors are only reported in INTERRUPT mode.

ERRORS:

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

Chapter 5

Input

CGI describes an input device as consisting of a ***measure*** and a list of ***associations*** with ***triggers***. A ***trigger*** corresponds to a physical input device such as a mouse button. A ***measure*** reports the current value of the device such as ***x,y*** position for a locator device. **SunCGI** input functions apply to all classes of input devices. The classes of devices that are offered are listed in Table 5-1.

Table 5-1: Devices Offered by SunCGI

Device Class	*Max. Number*	*Description*
IC_CHOICE	3	Mouse button (trigger)[17]
IC_LOCATOR	4	*x*, *y* position
IC_STRING	1	Keyboard input
IC_STROKE	3	Array of *x*, *y* positions
IC_VALUATOR	4	Normalized *x* position

When a trigger is activated, the measure of the device is reported to the application program by use of the event functions.

The effect of the event functions depends on the ***state*** of the input device (see Table 5-2). Before an input device is initialized it is in the RELEASED state. When an input device is first initialized, it is in the NO_EVENTS state.

The ***measure*** (value) of the input device is the device-dependent information that the application program wants (for example, the LOCATOR returns the x,y position of the cursor. The measure of an input device in the NO_EVENTS state is the measure that it was initialized with. The measure of an input device in the NO_EVENTS state does not change with activation of a trigger. For example, the measure of a LOCATOR ***does not*** change when the mouse moves.

The measure of an input device is changed by trigger activation when the input device is in either the REQUEST_EVENT, RESPOND_EVENT or QUEUE_EVENT state. If the input device is in the REQUEST_EVENT state its measure is set by the first trigger activation after the `initiate_request` function is called. If the input device is in the RESPOND_EVENT state its

[17] The left mouse button is choice 2, the middle button is choice 3, and so on.

measure is set by the most recent trigger activation. Previous trigger activations are not stored. When an input device is in the QUEUE_EVENT state, trigger activation is buffered by the ***event queue***. The ***event queue*** processes inputs in the order that they are received provided that the ***event queue*** is not ***overflowing***.

Table 5-2: States of Input Devices

State	*Brief Description*
RELEASED	Device not initialized.
NO_EVENTS	Device initialized, but unable to process events.
REQUEST_EVENT	Device measure set by first (and only first) trigger activation.
RESPOND_EVENT	Device measure settable by most recent trigger activation.
QUEUE_EVENT	Device able to post events on the event queue. Device measure settable by trigger activation when processed.

The figure below illustrates the CGI input state model.

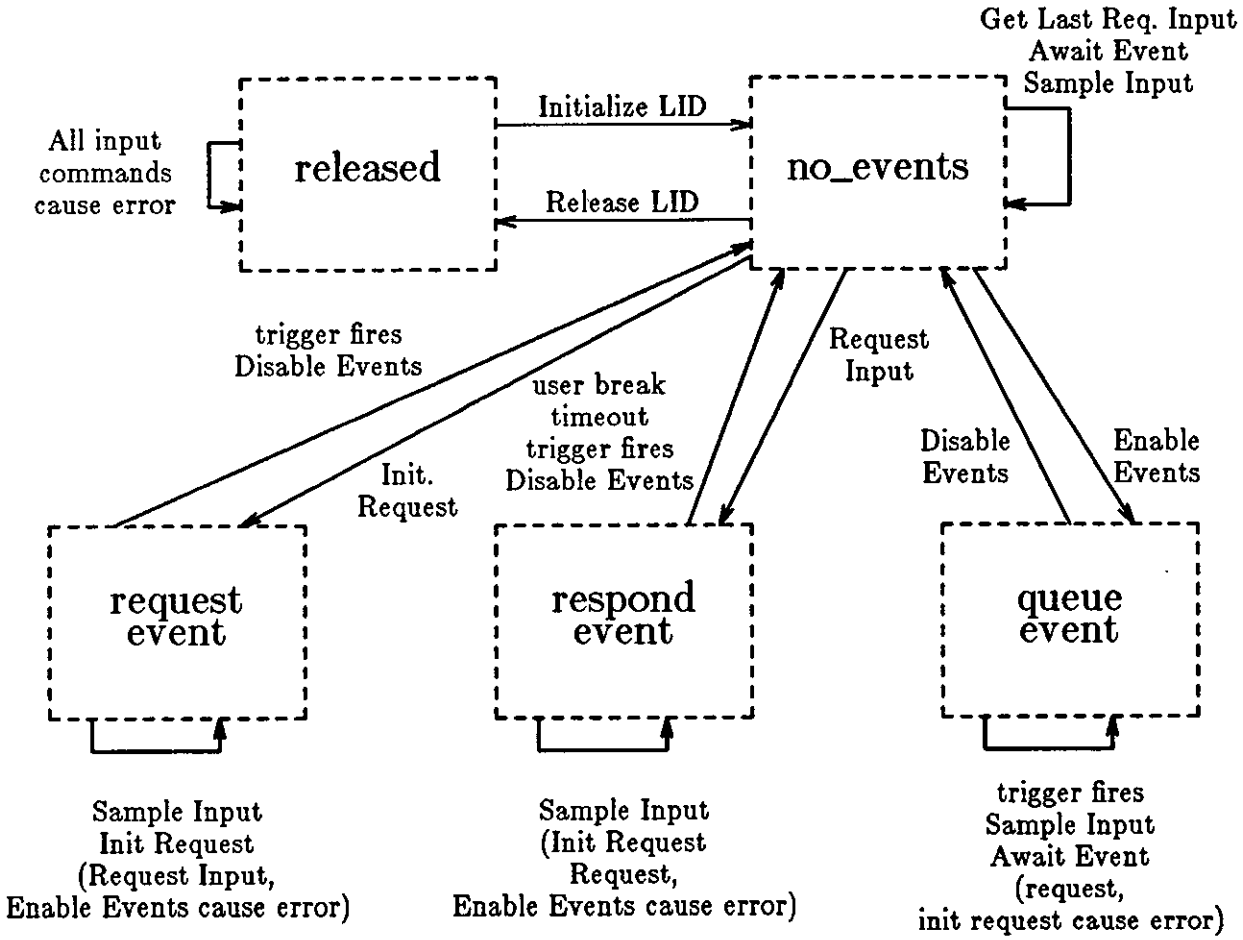


Figure 5-1: CGI Input State Model

In addition to reporting input to an application program, you may want to have the measure of a device displayed on all active view surfaces (called tracking). Tracking must be explicitly enabled for each device. In addition, the type of track is selectable for some input devices.

5.1. Input Device Management

Before input can be processed, the individual input devices must be initialized and associated with triggers. Input device initialization requires that at least one view surface is active. Typically, the procedure for initializing an input device includes calls to the `initialize_lid`, `enable_events`, and `associate` functions which turn on an input device and permit it to receive input from an associated trigger (see the following example).

```
{ /* initialize LOCATOR input device,  get input, and close */
Cinrep ival;
Clogical stat;
Cint trig;

ival.xypt.x = 16384; /* put cursor in the middle of the view surface  */
ival.xypt.y = 16384;
initialize_lid(IC_LOCATOR,1,&ival);
associate(2,IC_LOCATOR,1); /* associate locator with mouse button 1  */
request_input(IC_LOCATOR,1,5000000,&stat,&ival,&trig); /* wait five seconds */
if (stat == TRUE)
    printf(" trigger activated at %d %d \n",ival.xypt.x,ival.xypt.y);
else
    printf(" trigger not activated \n");
disable_events(IC_LOCATOR,1); /* shut device off */
dissociate(2,IC_LOCATOR,1);
release_input_device (IC_LOCATOR,1);
}
```

5.1.1. `initialize_lid` *(*`Cinitlid`*)*

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

`initialize_lid` (lid — Logical Input Device) must be called for an input device before it can be referenced by any other input function. ***initialize_lid*** changes the state of the specified input device from RELEASED to NO_EVENTS. The argument ***devclass*** specifies one of the supported devices, and ***devnum*** indicates the number of the device within that class. The argument ***ival*** sets the initial measure of the device. The structure `Cinrep` contains different elements for each type of measure. An error is generated if the device does not exist, if it is already initialized, or if the initial value is out of range. You must set the appropriate field of `ival`, 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 */
} Cinrep;
```

Notice that whenever a device is initialized, no associations with triggers are made. This must be done by having your application program call the appropriate functions.

ERRORS:

4	ENOTVSAC	CGI not in proper state: CGI shall be in state VSAC.
80	EINDNOEX	Input device does not exist.
82	EINDALIN	Input device already initialized.[18]
95	EBADDATA	Contents of input data record are invalid.
96	ESTRSIZE	Length of initial string is greater than the implementation defined maximum.

5.1.2. release_input_device *(Crelidev)*

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

release_input_device releases all associations with triggers, and removes all pending events from 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, devnum*** does not refer to an existing or initialized device.

ERRORS:

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

5.1.3. flush_event_queue *(Cflusheventqu)*

```
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). This function should be used

[18] 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.

carefully to avoid throwing away mouse-ahead or type-ahead inputs.

ERRORS:

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

5.1.4. selective_flush_of_event_queue *(*Cselectflusheventqu*)*

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

ERRORS:

5	ENOTOPOP	CGI not in proper state CGI shall be in either in state VDOP, VSOP, or VSAC.
80	EINDNOEX	Input device does not exist.
81	EINDINIT	Input device not initialized.

5.1.5. associate *(*Cassoc*)*

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

The function ***associate*** links a trigger to a specific device. The trigger numbers are listed in Table 5-3.

Table 5-3: Triggers Offered by SunCGI

Trigger	*Number*
Keyboard	1
Mouse Button No. 1	2
Mouse Button No. 2	3
Mouse Button No. 3	4
Mouse Position	5

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 explanation. IC_STROKE can only be linked with the mouse buttons. IC_STROKE returns an array of coordinates in VDC space. The first coordinate is entered when the mouse button is initially depressed, 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 ***depressed.***

ERRORS:

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

5.1.6. set_default_trigger_associations *(Csdefatrigassoc)*

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

The function ***set_default_trigger_associations*** links a trigger to a specific device. The rules for trigger association are the same as those for the ***associate*** function. The default associations are listed in Table 5-4.

Table 5-4: Default Trigger Associations

Device Class	*Default Trigger*
IC_LOCATOR	5
IC_VALUATOR	3
IC_STROKE	4
IC_CHOICE	2
IC_STRING	1

ERRORS:

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

5.1.7. dissociate (Cdissoc)

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

The function ***dissociate*** removes the association between a trigger and the specified device. If the ***dissociate*** function is called while there are events pending on the ***event queue***, the pending events are discarded.

ERRORS:

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

5.1.8. set_initial_value (Csinitval)

```
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 the 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.

ERRORS:

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

5.1.9. set_valuator_range (Csvalrange)

```
Cerror set_valuator_range(devnum, min, max)
    Cint devnum; /* device number */
    Cfloat min,max; /* limits of valuator */
```

`set_valuator_range` specifies the limits of the valuator. Device coordinates are mapped into the valuator range. All valuator events which are on the ***event queue*** are not rescaled. You must dequeue these events either with the `selective_flush_of_event_queue` function or `flush_event_queue`.

ERRORS:

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

5.2. Tracking

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 5-4). 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.

Table 5-5: Available Track Types

Trigger	*Number*	*Track Type*
IC_CHOICE	1.	PRINTERS_FIST
IC_LOCATOR	1.	PRINTERS_FIST
IC_STRING	1.	PRINTERS_FIST
	2.	STRING_TRACK
IC_STROKE	1.	PRINTERS_FIST
	2.	SOLID_LINE
	3.	X_LINE
	4.	Y_LINE
	5.	RUBBER_BAND_BOX
IC_VALUATOR	1.	PRINTERS_FIST
	2.	STRING_TRACK

5.2.1. `track_on` *(*`Ctrackon`*)*

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

track_on initiates track (or echo) for a specific device. The ***tracktype*** argument specifies the type of track to be used. See table 5-5. The ***trackregion*** argument specifies a rectangular subregion of the view surface (in VDC Space) where tracking is active. The returned argument ***value*** reports the device measure at the time ***track_on*** is called. The track is initially displayed on the first view surface opened.

The ***xypt*** element of ***value*** must be set to initially position the cursor. The reference point for IC_STROKE echos 2 through 5 is the first point in the stroke arrāy. The reference point for STRING_TRACK echo is the append_text concatenation point, and can be changed by calling text or append_text. The ***trackregion*** argument is not used and the device tracks in all areas of the view surface.

ERRORS:

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

5.2.2. track_off *(Ctrackoff)*

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

The function ***track_off*** terminates tracking for the specified input device. However, the printer's fist track always remains. For this reason, the ***tracktype*** and the ***action*** arguments are always ignored.

ERRORS:

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

5.3. Event Functions

Event functions allow the application program to set and obtain the current measure of input devices and triggers. No device-specific input routines exist in **SunCGI**. Therefore, each of the following functions requires explicit identification of an input device. There are two input buffers: the ***event queue*** and the ***request register***. The ***event queue*** is a FIFO (First In, First Out) buffer containing input events which are not generated by the ***request*** functions. The ***request register*** is a one-element per device buffer which contains the measure caused by the last input event, if the device has been put in ***request_event*** mode.

5.3.1. sample_input *(Csampinp)*

```
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 chapter introduction for an explanation of the relationship between the ***measure*** of an input device and the ***state*** of an input device.

ERRORS:

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

5.3.2. initiate_request *(Cinitreq)*

```
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. The value caused by the trigger activation can be obtained by the `get_last_requested_input` function.

ERRORS:

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

85 EINNTASD No triggers associated with device.

5.3.3. request_input (Creqinp)

```
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 awaits activation of a trigger associated with a specific device, for ***timeout*** microseconds. ***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. ***trigger*** is set to zero.

Cawresult is defined as an enumerated type as follows:

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

valid is set to VALID_DATA if a trigger is activated within the specified timeout period and TIMED_OUT otherwise. If the device is not in the state RESPOND_EVENT, ***valid*** is set to DISABLED. If the device is not a legal device, ***valid*** is set to NOT_SUPPORTED. ***valid*** is set to WRONG_STATE if **SunCGI** is not in state VSAC.

ERRORS:

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

5.3.4. get_last_requested_input (Cgetlastreqinp)

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

get_last_requested_input returns the contents of the ***request register***. The returned

argument ***valid*** reports if the device exists and is initialized. The returned argument ***sample*** reports the latest event in the ***request register***. If no event is in the request register, the initial device value is reported.

ERRORS:

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

5.3.5. enable_events *(*Cenevents*)*

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

Calling ***enable_events*** allows the specified device 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, devnum*** does not exist or is not initialized.

ERRORS:

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

5.3.6. disable_events *(*Cdaevents*)*

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

disable_events prevents the specified device from putting events on the ***event queue*** or making new associations with triggers. **disable_events** puts the input device in the NO_EVENT state. However, existing entries on the ***event queue*** are not removed and existing associations remain. ***devclass, devnum*** must refer to an existing or initialized device or an error is produced.

ERRORS:

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

5.3.7. await_event (Cawaitev)

```
Cerror await_event(timeout, valid, devclass, devnum,
        measure, message_link, replost, time_stamp, qstat, overflow)
    Cint timeout; /* amount of time to wait for input */
    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 */
```

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

If either the *event queue* is not empty or a trigger is activated, the class, the number, and the value of the device generating the event are reported in the returned arguments *devclass*, *devnum*, and *measure*. If *timeout* is less than 0, **SunCGI** waits until a trigger is activated. 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 *replost* and *time_stamp* arguments should be ignored and are always zero. The returned argument *qstat* reports the queue status after the 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* can assume one of two values: NO_OFLO, or OFLO.

ERRORS:

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

5.4. 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.

5.4.1. inquire_lid_state_list (Cqlidstatelis)

```
Cerror inquire_lid_state_list(devclass, devnum, valid, list)
    Cdevoff devclass;
    Cint devnum; /* device type, 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 RELEASE, 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:

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

5.4.2. inquire_lid_state (Cqlidstate)

```
Cerror inquire_lid_state(devclass, devnum, valid, state)
    Cdevoff devclass;
    Cint devnum; /* device type, 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:

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

80 EINDNOEX Input device does not exist.

5.4.3. inquire_trigger_state (Cqtrigstate)

```
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:

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

86 EINTRNEX Trigger does not exist.

5.4.4. inquire_event_queue_state (Cqevquestate)

```
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:

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

Appendix 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 three-dimensional 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 black-and-white 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*
Three-Dimensional 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

A.1.1. Output Aspects of SunCore not Supported by SunCGI

SunCGI does not support three-dimensional output primitives, current position, or textured color lookup tables for black-and-white devices. Since three-dimensional 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 black-and-white 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 four-by-four matrices, patterns in **SunCGI** have variable dimensions.

A.1.2. 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.

Appendix 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.

Unsupported Control Functions

```
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
```

Unsupported Input Functions

```
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.

Non Standard Control Functions

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

Non Standard Attribute Functions

```
define_bundle_index
line_endstyle
fixed_font
set_global_drawing_mode
```

Appendix C

Type Definitions

This appendix lists the types used by **SunCGI** functions. The definition of these types can be found in <cgitypes.h>. These definitions are listed here in alphabetical order to make the manual easier to read.

```
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    {
        CALID_DATA,
        TIMED_OUT,
        DISABLED,
        WRONG_STATE,
        NOT_SUPPORTED
} Cawresult;
```

```
typedef enum {
    BITTRUE, BITNOT
} 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    {
        CLIP,
        NOCLIP,
        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 {
    PRINTERS_FIST, NO_ECHO, 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 int     Cint;
```

```
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 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_TRUE, L_FALSE
} 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_ON, PROMPT_OFF
} 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 {
    FINAL, NOT_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;                           /* CGI tool descriptor */
} Cvwsurf;
```

Appendix D

Error Messages

This appendix lists the error messages in numerical and alphabetical 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)

0 NO_ERROR (0)
No error.

D.2. State Errors (1-5)

1 ENOTCGCL **CGI not in proper state: CGI shall 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*.

2 ENOTCGOP **CGI not in proper state: CGI shall 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* .

3 ENOTVSOP **CGI not in proper state: CGI shall 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*.

4 ENOTVSAC **CGI not in proper state: CGI shall 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*.

5 ENOTOPOP **CGI not in proper state CGI shall 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)

10 EVSIDINV **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.

11 ENOWSTYP **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.

12 EVSISOPN **Specified view surface is open.** An attempt was made to open a view surface which is already open. Corrective action involves removing one call to *open_vws*.

13 EVSNOTOP **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*.

14 EVSISACT **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*.

15 EVSNTACT **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*.

16 EINQALTL **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)

20 EBADRCTD **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.

21 EBDVIEWP **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.

22 ECLIPTOL **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.

23 ECLIPTOS **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*.

24 EVDCSDIL **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)

30 EBTBUNDL **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.

31 EBBDTBDI **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.

32 EBTUNDEF **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. Corrective action includes defining the entry by calling *define_attribute_environment_selector_index*.

33 EBADLINX **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.

34 EBDWIDTH **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.

35 ECINDXLZ **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.

36 EBADCOLX **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.

37 EBADMRKX **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.

38 EBADSIZE **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.

39 EBADFABX **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.

40 EPATARTL **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.

41 EPATSZTS **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.

42 ESTYLLEZ **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.

43 ENOPATNX **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.

44 EPATITOL **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.

45 EBADTXTX **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.

46 EBDCHRIX **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.

47 ETXTFLIN **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

48 ECEXFOOR **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).

49 ECHHTLEZ **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.

50 ECHRUPVZ **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.

51 ECOLRNGE **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)

60 ENMPTSTL **Number of points is too large.** The number of points exceeds 255. Change the ***n*** element of the **coorlist** structure to a value less than or equal to 255.

61 EPLMTWPT **polylines must have at least two points.** Change the ***n*** element of the **coorlist** structure to a value greater than 2 and add the corresponding points to the ***ptlist*** element.

62 EPLMTHPT **Polygons must have at least three points.** Change the ***n*** element of the **coorlist** structure to a value greater than or equal to 3 and add the corresponding points to the ***ptlist*** element.

63 EGPLISFL **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.

64 EARCPNCI **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);).

65 EARCPNEL **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).

66 ECELLATS **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.

67 ECELLPOS **Cell array dimensions must be positive.** Negative cell array dimensions are not permitted. Corrective action requires changing the parameters to the ***cell array*** function.

69 EVALOVWS **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***

70 EPXNOTCR **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)

80 EINDNOEX **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.

81 EINDINIT **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.

82 EINDALIN **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.

83 EINASAEX **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.

84 EINAIIMP **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.

85 EINNTASD **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.

86 EINTRNEX **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.

87 EINNECHO **Input device does not echo.** CHOICE devices do not support echo. Corrective action requires removing the call to ***echo_on*** from the application program.

88 EINECHON **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.

89 EINEINCP **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.

90 EINERVWS **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.

91 EINETNSU **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.

92 EINENOTO **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.

93 EIAEVNEN **Events already enabled.** Events have already been enabled for the specified device. The solution is to remove the offending call to ***enable_events.***

94 EINEVNEN **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.

95 EBADDATA **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.

96 ESTRSIZE **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.

97 EINQOVFL **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)

110 EMEMSPAC

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.

D.9. Possible Causes of Visual Errors

Table D-1: Possible Causes of Visual Errors

Behavior	*Possible Cause*
Segmentation fault for open_vws	a. devdd argument for open_vws is declared as a pointer (the address of devdd should be passed).
No primitives displayed	a. View surface not initialized. b. View surface not active. c. VDC to device coordinate mapping makes objects too small. d. Clipping rectangle is too small and clipping is ON. e. Perimeter visibility is set to OFF and interior style is set to HOLLOW. f. line_color or fill_color is set to background color.
Primitives displayed on undesired view surfaces	a. Undesired view surfaces have not been deactivated.
Segmentation fault for inquiry functions	a. passing variable instead of address (&) of variable.

Table D-2: Primitive-Specific Errors

Behavior	*Possible Cause*
Polylines or polymarkers aren't displayed.	a. Width or size is zero. b. Color is the same as background.
Polygon borders aren't displayed.	a. Width is zero. b. Color is the same as background. c. Perimeter visibility is set to OFF.
Circles aren't displayed.	a. Width or size is zero. b. Color is the same as background.
Ellipses aren't displayed.	a. Width or size is zero. b. Color is the same as background.
Text isn't displayed.	a. Width or size is zero. b. Color is the same as background. c. character height is too small. d. coordinates are outside the range of VDC Space or the clipping rectangle.
Cell arrays aren't displayed.	a. Dx or dy arguments are too small. b. Color is the same as background.
Cell arrays aren't displayed on all active view surfaces.	a. Mapping from cell size to view surface for smaller view surfaces is too small.
Pixel arrays aren't displayed.	a. Location is outside of view surface or clipping rectangle. b. Color is the same as background.
Bitblts aren't displayed.	a. Width or size is zero. b. Color is the same as background.

Table D-3: Attribute Errors

Behavior	*Possible Cause*
Attribute setting has no effect	a. attribute ASF is set to BUNDLED.
Text attributes have no effect	a. text precision is set to CHARACTER. b. attribute ASF is set to BUNDLED.
PATTERN fill is the same as HATCH	a. pattern_index and hatch_index are identical b. pattern_size is too small
PATTERN fill is different on different view surfaces.	a. View surfaces are of different size.

Table D-4: Input-specific Errors

Behavior	*Possible Cause*
Input device does not report	a. device not initialized
Input device does not echo	a. echo not initialized
Input device does not echo on whole view surface	a. echo region not set to whole view surface.

Appendix E

Sample Program

E.1. Martini Glass

The following program draws a Martini glass. The figure drawn by this program is identical to figure drawn by the example given in the appendix of the **SunCore** manual. It is suggested that novice users attempt to write this program to familiarize themselves with **SunCGI**.

```
#ifndef lint
static char sccsid[] = "@(#)glass.c 1.1 84/11/01 Copyr 1984 Sun Micro";
#endif

#include <cgidefs.h>

static Ccoorlist martinilist;
static Ccoor glass_coords[10] = {  0,0    ,
                                 -10,0    ,
                                  -1,1    ,
                                  -1,20   ,
                                 -15,35   ,
                                  15,35   ,
                                   1,20   ,
                                   1,1    ,
                                  10,0    ,
                                   0,0    };

static Ccoor water_coords[2] =  {-12,33   ,
                                  12,33   };

static Ccoor vpll = {-50,-10};
static Ccoor vpur = {50,80};


Cint name;

main()
{
   Cvwsurf device;


   device.dd=PIXWINDD;
   open_cgi();  /* initialize CGI */
   open_vws(&name,&device); /* open view surface */
   vdc_extent(&vpll,&vpur); /* reset VDC space */
   martinilist.ptlist=glass_coords; /* load polyline structure */
   martinilist.n=10;
   polyline(&martinilist); /* draw glass */
   martinilist.ptlist=water_coords;
   martinilist.n=2; /* draw waterline */
   polyline(&martinilist);
   sleep(10); /* display for 10 seconds */
   close_vws(name); /* exit */
   close_cgi();
   exit(0);
   }
```

Appendix F

Using SunCGI and Pixwins (Cgipw)

CGI and Pixwins calls can be integrated in a single application program by using the **Cgipw** functions. However, the CGI standard does not provide for facilities for dealing with multiple overlapping windows. Many users would like to have the richness of CGI's primitives combined with Pixwin's ability to automatically take care of multiple (potentially overlapping) windows.

> If you decide to use CGI and Pixwins, you **may not use the standard SunCGI calls**. Instead you should use **cgipw** calls. For example, `cgipw_polyline` should be used instead of `polyline`. Note that **cgipw** functions do not return error codes.

Using SunCGI and Pixwins involves using the basic CGI primitives to include a CGI pixwin descriptor (type `Ccgiwin`) as the first argument. The routines implementing the CGI standard output and attribute functions for **SunCGI** functions take a structure containing a specific pixwin and attribute pointer as their first argument. However, the file `<cgipw.h>` should be included instead of `<cgidefs.h>` in the application program. The four functions `open_pw_cgi`, `open_cgi_pw`, `close_cgi_pw`, and `close_pw_cgi` are necessary for managing the **SunCGI/*Pixwin*** interface.

F.1. open_pw_cgi

```
Cerror open_pw_cgi()
```

`open_pw_cgi` puts CGI in a known internal state by setting the attributes to the default values and setting the NDC to device coordinate mapping to 1:1. Therefore, unless the application program changes the mapping by explicitly calling CGI functions to reset the NDC to device coordinate mapping, 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. No standard errors are specified. If `open_pw_cgi` returns a nonzero result, then the initialization failed.

F.2. open_cgi_pw

```
Cerror open_cgi_pw(pw, desc, name)
    pixwin *pw; /* pixwin */
    Ccgiwin *desc; /* CGI pixwin descriptor */
    Cint name;
```

`open_cgi_pw` makes the pixwin pointed to by *pw* known to the internals of CGI. Calls to all CGI primitives will affect this pixwin. *Desc* is a pointer to a CGI pixwin which is used as the first argument to **cgipw** function. However, CGI does not guarantee that a pixwin exists or is any other way properly initialized. Calls may also be made to any pixwin function (see example program). Multiple calls to `open_cgi_pw` will result in primitives being displayed on multiple view surfaces. Attributes are local and apply only to the specified pixwins which have been opened by using `open_cgi_pw`.

F.3. close_cgi_pw

```
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.

F.4. close_pw_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.

F.5. Using Cgipw

After calling the two initialization functions (`open_pw_cgi`, `open_cgi_pw`) the user may call both pixwin and **SunCGI** primitives as specified in their respective manuals. Signals are not handled by **SunCGI** when it is used with pixwins. No error handling is done by **cgipw** functions — this makes **cgipw** more efficient but errors must be detected by the programmer. Therefore, the application program must insure that the NDC to device coordinate mapping is changed when the window size is changed. 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 called after the **SunCGI** call.

The **cgipw** functions is given in the table below. If one of the functions that you want to use is not listed, then you should use the normal **SunCGI** function. 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 the *desc* argument which is of type **Ccgiwin**. *Desc* is a pointer to pixwin descriptor obtained from the `open_cgi_pw` function.

F.6. List of Cgipw Functions

Table F-1: List of Cgipw Functions

CGI 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)
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)
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);

CGI Function Name	*Cgipw Function Name*
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)
pattern_table(index, m, n, colorind)	cgipw_pattern_table(desc, index, m, n, colorind)
perimeter_color(index)	cgipw_perimeter_color(desc, index)
perimeter_type(ttyp)	cgipw_perimeter_type(desc, ttyp)
perimeter_width(index)	cgipw_perimeter_width(desc, index)
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)
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)
track_off(devclass, devnum)	cgipw_track_off(desc, devclass, devnum)
track_on(devclass, tracktype, trackregion, value)	cgipw_track_on(desc, devclass, tracktype, trackregion, value)
vdm_text(c1, flag, tstring)	cgipw_vdm_text(desc, c1, flag, tstring)

F.7. Example Program

```
#include  <cgipw.h>
#include  <suntool/gfxsw.h>

struct  pixwin  *mypw;
struct  gfxsubwindow  *mine;

main()
{
        Cgiwin 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(name);
        close_pw_cgi();
}
```

Appendix G

Using SunCGI with Fortran-77 Programs

All functions provided in **SunCGI** may be called from FORTRAN-77 programs by linking them with the */usr/lib/libcgi77.a* library. This is done by using the *f77* compiler with a command line such as:

```
tutorial% f77 -o grab grab.f -lcgi77 -lcgi -lsunwindow -lpixrect -lm
```

where *grab.f* is the FORTRAN source program. Note that */usr/lib/libcgi.a* must be linked with the program (the **-lcgi** option), and */usr/lib/libcgi77.a* must come before 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-77, 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-77 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.
- 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** structures contain both `int`'s and `float`'s. For instance, the argument to `inquire_viewing_parameters` contains both `int`'s and `float`'s. This can be handled in FORTRAN by declaring a `real` array and an `integer` array and making them share storage by using an `equivalence` statement. Following the call to the inquiry function, the `real` components can be accessed using the `real` array and the `integer` components can be accessed using the `integer` array.
- 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.

G.2. Example Program

This example is the FORTRAN equivalent of the very simple program for drawing a martini glass.

```
      include 'cgidefs77.h'

      integer vsurf(VWSURFSIZE)
      integer name

      integer glassdx(9), glassdy(9)
      data glassdx /-10,9,0,-14,30,-14,0,9,-10/
      data glassdy /0,1,19,15,0,-15,-19,-1, 0/

      integer waterdx(9), waterdy(9)
      data waterdx /-10,9,0,-14,30,-14,0,9,-10/
      data waterdy /0,1,19,15,0,-15,-19,-1, 0/

c  initialize CGI
      call cfopencgi
c  open the view surface
      call cfopenvws(name, , PIXWINDD, , ,)
c  reset VDC space
      call cfvdcext(vpll, vpur)
c  drawr glass
      call cfpolyline(glassdx, glassdy, 10)
c  draw water surface
      call cfpolyline(waterdx, waterdy, 9)
c  display for 10 seconds
      call sleep(10)
c  close the view surface
      call cfclosevws(name)
c  close cgi and exit
      call cfclosecgi
c
      end
```

G.3. Correspondence Between C Names and FORTRAN Names

Correspondence Between C Names and FORTRAN Names

Long Name	*FORTRAN Equivalent*
activate_vws	cfactvws
append_text	cfaptext
associate	cfassoc
await_event	cfawaitev
bitblt_pattern_array	cfbtblpatarr
bitblt_patterned_source_array	cfbtblpatsouarr
bitblt_source_array	cfbtblsouarr
cell_array	cfcellarr
character_expansion_factor	cfcharexpfac
character_height	cfcharheight
character_orientation	cfcharorientation
character_path	cfcharpath
character_set_index	cfcharsetix
character_spacing	cfcharspacing
circle	cfcircle
circular_arc_3pt	cfcircarcthree
circular_arc_3pt_close	cfcircarcthreecl
circular_arc_center	cfcircarccent
circular_arc_center_close	cfcircarccentcl
clear_control	cfclrcont
clear_view_surface	cfclrvws
clip_indicator	cfclipind
clip_rectangle	cfcliprect
close_cgi	cfclosecgi
close_vws	cfclosevws
color_table	cfcotable
deactivate_vws	cfdeactvws
define_bundle_index	cfdefbundix
device_viewport	cfdevvpt
disable_events	cfdaevents
disjoint_polyline	cfdpolyline
dissociate	cfdissoc
echo_off	cfechooff
echo_on	cfechoon
echo_update	cfechoupd
ellipse	cfellipse
elliptical_arc	cfelliparc

Correspondence Between C Names and FORTRAN Names

Long Name	*FORTRAN Equivalent*
elliptical_arc_close	cfelliparccl
enable_events	cfenevents
fill_area_bundle_index	cfflareabundix
fill_color	cfflcolor
fixed_font	cffixedfont
flush_event_queue	cfflusheventqu
get_last_requested_input	cfgetlastreqinp
hard_reset	cfhardrst
hatch_index	cfhatchix
initialize_lid	cfinitlid
initiate_request	cfinitreq
inquire_aspect_source_flags	cfqasfs
inquire_bitblt_alignments	cfqbtblalign
inquire_cell_array	cfqcellarr
inquire_device_bitmap	cfqdevbtmp
inquire_device_class	cfqdevclass
inquire_device_identification	cfqdevid
inquire_drawing_mode	cfqdrawmode
inquire_event_queue	cfqevque
inquire_fill_area_attributes	cfqflareaatts
inquire_input_capabilities	cfqinpcaps
inquire_lid_capabilities	cfqlidcaps
inquire_lid_state_list	cfqlidstate
inquire_lid_state_list	cfqlidstatelis
inquire_line_attributes	cfqlnatts
inquire_marker_attributes	cfqmkatts
inquire_output_capabilities	cfqoutcap
inquire_output_function_set	cfqoutfunset
inquire_pattern_attributes	cfqpatatts
inquire_physical_coordinate_system	cfqphyscsys
inquire_pixel_array	cfqpixarr
inquire_text_attributes	cfqtextatts
inquire_text_extent	cfqtextext
inquire_trigger_capabilities	cfqtrigcaps
inquire_trigger_state	cfqtrigstate
inquire_vdc_type	cfqvdctype
interior_style	cfintstyle
line_color	cflncolor
line_endstyle	cflnendstyle

Correspondence Between C Names and FORTRAN Names

Long Name	*FORTRAN Equivalent*
line_type	cflntype
line_width	cflnwidth
line_width_specification_mode	cflnwidthspecmode
marker_color	cfmkcolor
marker_size	cfmksize
marker_size_specification_mode	cfmksizespecmode
marker_type	cfmktype
open_cgi	cfopencgi
open_vws	cfopenvws
partial_polygon	cfppolygon
pattern_index	cfpatix
pattern_reference_point	cfpatrefpt
pattern_size	cfpatsize
pattern_table	cfpattable
perimeter_color	cfperimcolor
perimeter_type	cfperimtype
perimeter_width	cfperimwidth
perimeter_width_specification_mode	cfperimwidthspecmode
pixel_array	cfpixarr
polygon	cfpolygon
polyline	cfpolyline
polyline_bundle_index	cfpolylnbundix
polymarker	cfpolymarker
polymarker_bundle_index	cfpolymkbundix
rectangle	cfrectangle
release_input_device	cfrelidev
request_input	cfreqinp
reset_to_defaults	cfrsttodefs
sample_input	cfsampinp
selective_flush_of_event_queue	cfselectflusheventqu
set_aspect_source_flags	cfsaspsouflags
set_default_trigger_associations	cfsdefatrigassoc
set_drawing_mode	cfsdrawmode
set_error_warning_mask	cfserrwarnmk
set_global_drawing_mode	cfsgldrawmode
set_initial_value	cfsinitval
set_up_sigwinch	cfsupsig
set_valuator_range	cfsvalrange
text	cftext

Correspondence Between C Names and FORTRAN Names

Long Name	*FORTRAN Equivalent*
text_alignment	cftextalign
text_bundle_index	cftextbundix
text_color	cftextcolor
text_font_index	cftextfontix
text_precision	cftextprec
vdc_extent	cfvdcext
vdm_text	cfvdmtext

G.4. 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.

```
integer function cfactvws(name)
    integer   name

integer function cfaptext(flag, string, stringlen)
    integer   flag
    character*(*) string
    integer   stringlen

integer function cfassoc(trigger, devclass, devnum)
    integer   trigger
    integer   devclass
    integer   devnum
```

```
integer function cfawaitev(timeout, valid, devclass, devnum,
1       x, y, xlist, ylist, n, val, choice, string, stringlen,
2       message_link, replost, time_stamp, 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   stringlen
    integer   message_link
    integer   replost
    integer   time_stamp
    integer   qstat
    integer   overflow

integer function cfbtblpatarr(pixpat, px, py, pixtarget,
1       rx, ry, ox, oy, dx, dy)
    integer   pixpat
    integer   px, py
    integer   pixtarget
    integer   rx, ry
    integer   ox, oy
    integer   dx, dy

integer function cfbtblpatsouarr(pixpat, px, py, pixsource,
1       sx, sy, pixtarget, rx, ry, ox, oy, dx, dy)
    integer   pixpat
    integer   px, py
    integer   pixsource
    integer   sx, sy
    integer   pixtarget
    integer   rx, ry
    integer   ox, oy
    integer   dx, dy

integer function cfbtblsouarr(bitsource, xo, yo, xe, ye, bittarget, xt, yt)
    integer   bitsource, bittarget
    integer   xo, yo, xe, ye, xt, yt

integer function cfcellarr(px, qx, rx, py, qy, ry, dx, dy, colorind)
    integer   px, py
    integer   qx, qy
    integer   rx, ry
    integer   dx, dy
    integer   colorind(*)
```

```
integer function cfcharexpfac(efac)
    real        efac

integer function cfcharheight(height)
    integer   height

integer function cfcharorient(bx, by, dx, dy)
    real        bx, by, dx, dy

integer function cfcharpath(path)
    integer   path

integer function cfcharsetix(index)
    integer   index

integer function cfcharspacing(efac)
    real        efac

integer function cfcircarccent(c1x, c1y, c2x, c2y, c3x, c3y, rad)
    integer   c1x, c1y, c2x, c2y, c3x, c3y
    integer   rad

integer function cfcircarccentcl(c1x, c1y, c2x, c2y, c3x, c3y, rad, close)
    integer   c1x, c1y, c2x, c2y, c3x, c3y
    integer   rad
    integer   close

integer function cfcircarcthree(c1x, c1y,  c2x, c2y, c3x, c3y)
    integer   c1x, c1y,  c2x, c2y, c3x, c3y

integer function cfcircarcthreecl(c1x, c1y,  c2x, c2y, c3x, c3y, close)
    integer   c1x, c1y,  c2x, c2y, c3x, c3y
    integer   close

integer function cfcircle(x, y, rad)
    integer   x
    integer   y
    integer   rad

integer function cfclipind(flag)
    integer   flag

integer function cfcliprect(xmin, xmax, ymin, ymax)
    integer   xmin, xmax, ymin, ymax

integer function cfclosecgi()
```

```
integer function cfclrcont(soft, hard, intern, extent)
    integer   soft, hard
    integer   intern
    integer   extent

integer function cfclrvws(name, defflag, color)
    integer   name
    integer   defflag
    integer   color

integer function cfcotable(istart, ra, ga, ba, n)
    integer   istart
    integer   ra(*), ga(*), ba(*)
    integer   n

integer function cfdaevents(devclass, devnum)
    integer   devclass
    integer   devnum

integer function cfdeactvws(name)
    integer   name

integer function cfdevvpt(name, xbot, ybot, xtop, ytop)
    integer   name
    integer   xbot, ybot, xtop, ytop

integer function cfdissoc(trigger, devclass, devnum)
    integer   trigger
    integer   devclass
    integer   devnum

integer function cfdpolyline(xcoors, ycoors, n)
    integer   xcoors
    integer   ycoors
    integer   n

integer function cfechooff(devclass, devnum, echo)
    integer   devclass
    integer   devnum
    integer   echo
```

```
integer function cfechoon(devclass, echotype, exlow, eylow,
1       exup, eyup, x, y, xlist, ylist, n, val, choice, string, stringlen)
    integer   devclass
    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   stringlen

integer function cfechoupd(echo, x, y, xlist, ylist, n, val,
1       choice, string, stringlen)
    integer   echo
    integer   x, y
    integer   xlist(*)
    integer   ylist(*)
    integer   n
    real      val
    integer   choice
    character*(*) string
    integer   stringlen

integer function cfelliparc(x, y , sx, sy, ex, ey, majx, miny)
    integer   x, y
    integer   sx, sy
    integer   ex, ey
    integer   majx, miny

integer function cfelliparccl(x, y , sx, sy, ex, ey, majx, miny, close)
    integer   x, y
    integer   sx, sy
    integer   ex, ey
    integer   majx, miny
    integer   close

integer function cfellipse(x, y, majx, miny)
    integer   x, y
    integer   majx, miny

integer function cfenevents(devclass, devnum)
    integer   devclass
    integer   devnum
```

```
integer function cffixedfont(index)
    integer   index

integer function cfflareabundix(index)
    integer   index

integer function cfflcolor(color)
    integer   color

integer function cfflusheventqu()

integer function cfgetlastreqinp(devclass, devnum, valid,
1       x, y, xlist, ylist, n, val, choice, string, stringlen)
    integer   devclass
    integer   devnum
    integer   valid
    integer   x, y
    integer   xlist(*)
    integer   ylist(*)
    integer   n
    real      val
    integer   choice
    character*(*) string
    integer   stringlen

integer function cfhardrst()

integer function cfhatchix(index)
    integer   index

integer function cfinitlid(devclass, devnum, x, y, xlist, ylist, n, val,
1       choice, string, stringlen)
    integer   devclass
    integer   devnum
    integer   x, y
    integer   xlist(*)
    integer   ylist(*)
    integer   n
    real      val
    integer   choice
    character*(*) string
    integer   stringlen

integer function cfinitreq(devclass, devnum)
    integer   devclass
    integer   devnum
```

```
integer function cfintstyle(istyle, perimvis)
    integer   istyle
    integer   perimvis

integer function cflncolor(index)
    integer   index

integer function cflnendstyle(ttyp)
    integer   ttyp

integer function cflntype(ttyp)
    integer   ttyp

integer function cflnwidth(index)
    real        index

integer function cflnwidthspecmode(mode)
    integer   mode

integer function cfmkcolor(index)
    integer   index

integer function cfmksize(index)
    real        index

integer function cfmksizespecmode(mode)
    integer   mode

integer function cfmktype(ttyp)
    integer   ttyp

integer function cfopencgi()
```

```
integer function cfopenvws(name, screenname, screenlen,
1       windowname, windowlen, windowfd, retained, dd,
2       cmapsize, cmapname, cmaplen, flags, ptr, noargs)
    integer   name
    character*(*) screenname
    integer   screenlen
    character*(*) windowname
    integer   windowlen
    integer   windowfd
    integer   retained
    integer   dd
    integer   cmapsize
    integer   cmapname(*)
    integer   cmaplen
    integer   flags
    character*(*) ptr
    integer   noargs


integer function cfpatix(index)
    integer   index


integer function cfpatrefpt(x, y)
    integer   x, y


integer function cfpatsize(dx, dy)
    integer   dx, dy


integer function cfpattable(index, m, n, colorind)
    integer   index
    integer   m, n
    integer   colorind


integer function cfperimcolor(index)
    integer   index


integer function cfperimtype(ttyp)
    integer   ttyp


integer function cfperimwidth(index)
    real        index


integer function cfpixarr(px, py, m, n, colorind)
    integer   px, py
    integer   m, n
    integer   colorind(*)
```

```
integer function cfpolygon(xcoors, ycoors, n)
    integer   xcoors
    integer   ycoors
    integer   n

integer function cfpolyline(xcoors, ycoors, n)
    integer   xcoors
    integer   ycoors
    integer   n

integer function cfpolylnbundix(index)
    integer   index

integer function cfpolymarker(xcoors, ycoors, n)
    integer   xcoors
    integer   ycoors
    integer   n

integer function cfpolymkbundix(index)
    integer   index

integer function cfppolyline(xcoors, ycoors, n, flag)
    integer   flag

integer function cfqasfs(n, num, vals)
    integer   n
    integer   num(*)
    integer   vals(*)

integer function cfqcellarr(px, qx, rx, py, qy, ry, dx, dy, colorind)
    integer   px, py
    integer   qx, qy
    integer   rx, ry
    integer   dx, dy
    integer   colorind(*)

integer function cfqdevbtmp(name, map)
    integer   name, map

integer function cfqdevclass(output, input)
    integer   output, input

integer function cfqdevid(name, devid, stringlen)
    integer   name
    character*(*) devid
    integer   stringlen
```

```
integer function cfqflareaatts(style, vis, hindex, pindex, bindex,
1       pstyle, pwidth, pcolor)
    integer   style, vis, hindex, pindex
    integer   bindex
    integer   pstyle
    real      pwidth
    integer   pcolor

integer function cfqinpcaps(valid, numval, numstrk, numchoice,
1       numstr, numtrig, evqueue, asynch, coordmap, tracking,
2       prompt, acknowledgement, trigman)
    integer   valid
    integer   numval
    integer   numstrk
    integer   numchoice
    integer   numstr
    integer   numtrig
    integer   evqueue
    integer   asynch
    integer   coordmap
    integer   tracking
    integer   prompt
    integer   acknowledgement
    integer   trigman

integer function cfqlidcaps(devclass, devnum, valid, sample,
1       change,   numassoc,  trigassoc, prompt, acknowledgement,
2       echo, classdep, stringlen, state)
    integer   devclass
    integer   devnum
    integer   valid
    integer   sample
    integer   change
    integer   numassoc
    integer   trigassoc(*)
    integer   prompt
    integer   acknowledgement
    integer   echo(*)
    character*(*) classdep
    integer   stringlen
    integer   state(*)
```

```
integer function cfqlidstatelis(devclass, devnum, valid, numloc,
1       numval, numstrk, numchoice, numstr, numtrig, eventqueue,
2       asynch, coordmap, echo, tracking, prompt, acknowledgement,
3       triggermanipulation)
    integer   devclass
    integer   devnum
    integer   valid
    integer   numloc
    integer   numval
    integer   numstrk
    integer   numchoice
    integer   numstr
    integer   numtrig
    integer   eventqueue
    integer   asynch
    integer   coordmap
    integer   echo
    integer   tracking
    integer   prompt
    integer   acknowledgement
    integer   triggermanipulation

integer function cfqlnatts(style, width, color, index)
    integer   style
    real      width
    integer   color, index

integer function cfqmkatts(type, size, color, index)
    integer   type
    real      size
    integer   color, index

integer function cfqoutcap(first, last, list)
    integer   first, last
    character*(*) list

integer function cfqoutfunset(level, support)
    integer   level
    integer   support

integer function cfqpatatts(cindex, row, column, colorlis, x, y, dx, dy)

integer function cfqphyscsys(name, xbase, ybase, xext, yext, xunits, yunits)
    integer   name
    integer   xbase, ybase
    integer   xext, yext
    real       xunits, yunits
```

```
integer function cfqtext(fontset, index, cfont, prec, efac, space,
1       color, hgt, bx, by, ux, uy, path, halign, 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


integer function cfqtextext(string, stringlen, nextchar,
1       conx, cony, llpx, llpy, ulpx, ulpy, urpx, urpy)
    character*(*) string
    character*(*) nextchar
    integer   stringlen
    integer   conx
    integer   cony
    integer   llpx
    integer   llpy
    integer   ulpx
    integer   ulpy
    integer   urpx
    integer   urpy


integer function cfqtrigcaps(trigger, valid, change, n, class,
1       assoc, numassoc, maxassoc, prompt,     acknowledgement,
1         name, namelen,   description, desclen)
    integer   trigger
    integer   valid
    integer   change
    integer   n
    integer   class
    integer   assoc(*)
    integer   numassoc
    integer   maxassoc
    integer   prompt
    integer   acknowledgement
    character*(*) name
    integer   namelen
    character*(*) description
    integer   desclen


integer function cfqtrigstate(trigger, valid, state, n, class, assoc)
    integer   trigger
    integer   valid
    integer   state
    integer   n
    integer   class(*)
    integer   assoc(*)


integer function cfqvdctype(type)
    integer   type
```

```
integer function cfrectangle(xbot, ybot, xtop, ytop)
    integer   xbot, ybot, xtop, ytop

integer function cfrelidev(devclass, devnum)
    integer   devclass
    integer   devnum

integer function cfreqinp(devclass, devnum, timeout, valid, sample,
1       trigger, x, y, xlist, ylist, n, val, choice, string, stringlen)
    integer   devclass
    integer   devnum
    integer   timeout
    integer   valid
    integer   x, y
    integer   xlist(*)
    integer   ylist(*)
    integer   n
    real      val
    integer   choice
    character*(*) string
    integer   stringlen
    integer   trigger

integer function cfrsttodefs(name)
    integer   name

integer function cfsampinp(devclass, devnum, valid, x, y,
1       xlist, ylist, n, val, choice, string, stringlen)
    integer   devclass
    integer   devnum
    integer   valid
    integer   x, y
    integer   xlist(*)
    integer   ylist(*)
    integer   n
    real      val
    integer   choice
    character*(*) string
    integer   stringlen

integer function cfsaspsouflags(fval, fnum, n)
    integer   fval(*), fnum(*), n

integer function cfsdefatrigassoc(devclass, devnum)
    integer   devclass
    integer   devnum
```

```
integer function cfsdrawmode(visibility, source, destination, combination)
    integer   visibility
    integer   source
    integer   destination
    integer   combination

integer function cfselectflushevenqu(devclass, devnum)
    integer   devclass
    integer   devnum

integer function cfserrwarnmk(action)
    integer   action

integer function cfsgldrawmode(combination)
    integer   combination

integer function cfsinitval(devclass, devnum, x, y, xlist, ylist, n,
1       val, choice, string, stringlen)
    integer   devclass
    integer   devnum
    integer   x, y
    integer   xlist(*)
    integer   ylist(*)
    integer   n
    real      val
    integer   choice
    character*(*) string
    integer   stringlen

integer function cfsupsig(sig_function)
    integer   sig_function
    external sig_function

integer function cfsvalrange(devnum, mn, mx)
    integer   devnum
    real      mn, mx

integer function cftext(x, y, string, stringlen)
    integer   x
    integer   y
    character*(*) string
    integer   stringlen

integer function cftextalign(halign, valign, hcalind, vcalind)
    integer   halign
    integer   valign
    real      hcalind, vcalind
```

```
integer function cftextbundix(index)
    integer   index

integer function cftextcolor(index)
    integer   index

integer function cftextfontix(index)
    integer   index

integer function cftextprec(ttyp)
    integer   ttyp

integer function cfvdcext(xbot, ybot, xtop, ytop)
    integer   xbot, ybot, xtop, ytop

integer function cfvdmtext(x, y, flag, string, stringlen)
    integer   x
    integer   y
    integer   flag
    character*(*) string
    integer   stringlen

integer function vqdrawmode(visibility, source, destination, combination)
    integer   visibility
    integer   source
    integer   destination
    integer   combination

integer function vqfpixarr(px, py, m, n, colorind)
    integer   px, py
    integer   m, n
    integer   colorind(*)
```

Index

D

E

F

G

H

I

L

R

S

T

U

V

W

X