

The SCO CGI®
Development System

Release Notes

Version 1.0

The Santa Cruz Operation, Inc.

1

1

Information in this document is subject to change without notice and does not represent a commitment on the part of The Santa Cruz Operation, Inc. nor Microsoft Corporation. The software described in this document is furnished under a license agreement or nondisclosure agreement. The software may be used or copied only in accordance with the terms of the agreement. It is against the law to copy this software on magnetic tape, disk, or any other medium for any purpose other than the purchaser's personal use.

ALL USE, DUPLICATION, OR DISCLOSURE WHATSOEVER BY THE U.S. GOVERNMENT SHALL BE EXPRESSLY SUBJECT TO RESTRICTIONS AS SET FORTH IN SUBDIVISION (b) (3) (ii) FOR RESTRICTED RIGHTS IN COMPUTER SOFTWARE AND SUBDIVISION (b) (2) FOR LIMITED RIGHTS IN TECHNICAL DATA, BOTH AS SET FORTH IN FAR 52.227-7013.

Portions ©1986, 1987 Graphic Software Systems, Inc.

All rights reserved.

Portions ©1987 The Santa Cruz Operation, Inc.

All rights reserved.

This document was typeset with an IMAGEN® 8/300 Laser Printer.

Microsoft and XENIX are registered trademarks of Microsoft Corporation.
IMAGEN is a registered trademark of IMAGEN Corporation.
SCO CGI is a trademark of The Santa Cruz Operation, Inc.
SoftCare is a service mark of The Santa Cruz Operation, Inc.
Apple LaserWriter is a trademark of Apple Computer, Inc.
HP Plotter, HP Laserjet, and Thinkjet are registered trademarks of Hewlett-Packard Company.
IBM EGA and IBM CGA are trademarks of International Business Machines, Inc.
Epson is a registered trademark of Epson America, Inc.
GSS and the GSS logo are registered trademarks, and GSS*CGI and GSS*GRA FTERM are trademarks of Graphic Software Systems, Incorporated.

SCO Document Number: CGI/286/386-11-23-87-1.0.1

Processed Date: Wed Nov 25 10:25:03 PST 1987

SCO CGI Development System Release Notes

1. Preface 1
2. Your Software Package 2
 - 2.1 SoftCare Support 2
3. Supported Environments 3
4. Disk Usage 3
5. A Note to Developers of XENIX Graphics Applications 3
6. Installing SCO CGI 5
 - 6.1 Using Custom 5
 - 6.2 The SCO CGI Files 9
 - 6.3 Packages in This Set 10
7. Programmer's Guide Notes 10
 - 7.1 Signals 10
8. Setting Environmental Variables 11
9. Compiling and Running the Test Program 17
 - 9.1 Compiling Procedure 18
 - 9.2 Test Program Frame Descriptions 19
 - 9.2.1 Frame 1 19
 - 9.2.2 Frame 2 21

9.2.3	Frame 3	22
9.2.4	Frame 4	22
9.2.5	Frame 5	22
9.2.6	Frame 6	23
9.2.7	Frame 7	23
9.2.8	Frame 8	25
9.2.9	Frame 9	25
9.2.10	Frame 10	26
9.2.11	Frame 11	27
9.2.12	Frame 12	27
9.2.13	Frame 13	27
9.2.14	Frames 14-16	28
9.2.15	Frame 17	29
9.2.16	Frame 18	29
9.2.17	Frame 19	30
9.2.18	Frame 20	30

10. Known Bugs and Omissions in This Release 31

Release and Installation Notes
Release 1.0
SCO Computer Graphics Interface
for Personal Computers
Development System
November 23, 1987

1. Preface

Thank you for purchasing the SCO XENIX Development System with the SCO Computer Graphics Interface for personal computers.

These release notes contain information on the software and documentation, plus instructions for installing the SCO CGI software on your system. They describe selecting the options appropriate to your configuration, the distribution medium which makes up the system, interfacing application programs with SCO CGI.

SCO CGI is a software library of graphics subroutines and device drivers that enables you to develop device-independent application programs.

The subroutines are specified in Chapter 3 of the *SCO CGI Programmer's Guide*, *SCO CGI Functions* and in the *C Language Reference Guide*. To use SCO CGI with a graphics application program, include the subroutine calls in your source code and compile as usual. See the *C Language Reference Guide* for information on linking your application program with the desired language-binding library.

We always appreciate hearing about users' experience with our product, as well as their recommendations for making it even more useful. All written suggestions by user's are given serious consideration.

2. Your Software Package

Your SCO CGI software package includes the following items:

- *These Release and Installation Notes.*
- *The SCO CGI Getting Started, Tutorial, User's Guide, Configuration Guide, and Quick Reference Guide.*
- **The distribution media that you use to install SCO CGI on your hard disk.**
- **A software license agreement.**
- **A serialization card that contains your serial number and activation key, both of which are needed to install SCO CGI.**
- **A Customer Registration Form, which you should complete and return to us within five days of receiving your software package. To be eligible for our support services, please supply all information requested on the form.**
- **A SoftCare Support Agreement Registration Form (US and Canada only). Complete this form if you want SoftCare support beyond the first month.**

2.1 SoftCare Support

SoftCare support is available to customers who purchased SCO CGI for use in the United States and Canada. If you purchased SCO CGI for use in other countries, contact your distributor or retailer for support information.

You can choose from several levels of support. The available levels of support are described on an insert at the back of the SCO CGI documentation. The amount of your payment indicates which level of support you want. If you choose Level II Support, be sure to include on the card the name of the person you appoint as the support contact, the one authorized to call our SoftCare hotline.

3. Supported Environments

SCO CGI supports the following operating systems and media types:

386 Operating Systems	Media
SCO XENIX 386 System V	3½" DSDD
SCO XENIX 386 System V	5¼" DSDD
SCO XENIX 386 System V	5¼" DSDD

286 Operating Systems	Media
SCO XENIX 286 System V	3½" DSDD
SCO XENIX 286 System V	5¼" DSDD
SCO XENIX 286 System V	5¼" DSDD

4. Disk Usage

The contents of this distribution consume approximately 2655 Kbytes of disk space. Most files are extracted into directories beginning with */usr*. Be certain that the necessary space is available on the file systems before attempting installation. The following table shows you how much space you need on the file systems:

Filesystem	Usage
<i>/usr</i>	2560 KBytes
<i>/tmp</i>	95 KBytes

5. A Note to Developers of XENIX Graphics Applications

Continuing in our tradition of expanding XENIX environments, The Santa Cruz Operation has adapted the popular GSS*CGI[™] to work closely with SCO XENIX. At this time, the SCO CGI product is

included with the SCO XENIX Development System at no extra charge. With the SCO CGI, developers now have the tools to create high-quality graphics applications packages for the XENIX environment.

Included with the SCO CGI package, also at no additional cost, are selected graphics device drivers* that developers can use to test the functionality of their graphics applications on various peripherals. These drivers are not, however, for further distribution without the prior consent of The Santa Cruz Operation, Inc.

SCO offers the SCO Graphics Run Time Package to run graphics applications developed with the SCO CGI. Please call The Santa Cruz Operation regarding the availability of the the SCO Graphics Run Time Package, as well as for additional information.

* The graphics device drivers included are:

Apple LaserWriter	HP Plotter
Enhanced Graphics Adapter (EGA)	HP Laserjet
Color Graphics Adapter (CGA)	HP Thinkjet
Epson MX/FX 80/100 printers	GSS Grafterm
SCO Metafile	

Note

The EGA driver */usr/lib/cgi/ega* and the CGA High-Resolution driver */usr/lib/cgi/cgabw* are supported in this release of SCO CGI.

6. Installing SCO CGI

Before you can install SCO CGI on your hard disk, you will need the following items:

- The SCO CGI distribution media.
- The serial number and activation key, which are alphanumeric codes located on your serialization card.

6.1 Using Custom

You must be logged in as *root* (the “super user”) to install the SCO CGI software. The super user has access to all the system files and, as such, should take care to preserve them.

If you are not running SCO XENIX System V Release 2.2.2 or later, you also need the SCO XENIX System V Link Kit installed prior to installation of the EGA driver.

1. To move to the */* directory, type the following command and press <Return>.

```
cd /
```

2. Login as *root*, type *custom* and then press <Return>. You see:

```
1. Operating System
2. Development System
3. Text Processing System
4. Add a Supported Product
```

3. Select option 4 – Add a supported product (SCO CGI).
4. You are prompted to insert the first volume of the distribution media. Insert volume 1 and press <Return>.

5. The system displays:

Installing custom data file...

Insert distribution volume 1
and press <Return> or enter q to quit.

6. The system displays the following menu:

1. Install one or more packages
2. Remove one or more packages
3. List the files in a package
4. Install a single file
5. Select a new set to customize
6. Display current disk usage
7. Help

7. Choose option 1 – Install – to install any of the packages in the product.
8. The packages contained in SCO CGI are displayed on your screen .
9. You are prompted to enter the packages you wish to install. Enter them or type **all**, for all of the SCO CGI packages. Press <Return> .

If you select the EGA package and you are not running SCO XENIX System V Release 2.2.2 or later, you need to re-link the kernel. This is also necessary if you select **all**, because the EGA package is included with the other SCO CGI packages. Re-linking the kernel is described later on in this installation procedure.

10. You see:

Insert SCO Computer Graphics Interface volume 1
and press <Return> or enter q to quit:

Volume 1 should still be in the drive. Press <Return>. You
see:

Extracting Files...

If the product has more than one volume, you are asked to
load each additional volume in turn.

11. After all the files are extracted, a restricted-rights legend is
displayed on your screen.
12. You are prompted to enter your serial number.
The system displays:

SCOCGI serialization

Enter your serial number or enter q to quit:

Enter your serial number exactly as it appears on the
serialization card, including the three-letter prefix. Press
<Return>.

13. The system then displays:

Enter your activation key or enter q to quit:

Enter your activation key exactly as it appears on the serialization card. Press <Return>. If you mistype your activation key, you are prompted to enter it again.

14. If the Link Kit is not already installed, the system prompts you to install it now. You see:

The Link Kit is only partially installed.
Do you wish to install it now? (y/n)

If you enter **y**, the system prompts you to insert the proper operating system distribution floppy and proceed with the installation of the Link Kit and SCO CGI. If you enter **n**, SCO CGI installation fails and returns you to the custom menu. You must then install the Link Kit and reinstall SCO CGI.

15. After you enter your activation key and the Link Kit is properly installed, you see:

Do you wish to create a new kernel? (y/n)

Enter **y** to create a new kernel and you see:

Re-linking the kernel...

The system displays a message indicating that a new kernel is being made and that file permissions are being checked.

16. You are returned to the custom menu. Enter **q** to have a system prompt returned to you. Remember to remove the

distribution media from the disk drive.

17. If you have installed the EGA package, reboot the system using `/etc/shutdown`. Refer to the *SCO XENIX System V Installation Guide* for further instructions on rebooting your system.

For the safety of your files, do not use SCO CGI when logged into the *root* account. File permissions protect you from unintentionally overwriting certain files when using SCO CGI as a normal user. However, when logged into the *root* account, you can accidentally overwrite any file. Therefore, log out of the *root* account before you actually use SCO CGI.

SCO CGI is now installed and ready to use. To begin working with SCO CGI, log into a user account and refer to the SCO CGI documentation for further instructions.

6.2 The SCO CGI Files

The SCO CGI files on your distribution media are listed below.

<code>./usr/lib/cgi/cgabw</code>	<code>./usr/lib/cgi/cgaco</code>
<code>./usr/lib/cgi/ega</code>	<code>./usr/lib/cgi/iaserwriter</code>
<code>./usr/lib/cgi/epson80</code>	<code>./usr/lib/cgi/epson100</code>
<code>./usr/lib/cgi/hpplot</code>	<code>./usr/lib/cgi/laserjet</code>
<code>./usr/lib/cgi/cgmdd</code>	<code>./usr/lib/cgi/grftrmdd</code>
<code>./usr/lib/cgi/thinkjet</code>	<code>./usr/lib/cgi/cgiprep</code>
<code>./usr/lib/cgi/sample/cgitest.c</code>	<code>./usr/lib/cgi/sample/makefile</code>
<code>./usr/lib/Llibccgi.a</code>	<code>./usr/lib/Mlibccgi.a</code>
<code>./usr/lib/Slibccgi.a</code>	<code>./usr/lib/386/Slibccgi.a</code>
<code>./usr/lib/cgi/fonts/fontlist.dat</code>	
<code>./usr/lib/cgi/fonts/ibmbw.bld</code>	<code>./usr/lib/cgi/fonts/ibmbw.mon</code>
<code>./usr/lib/cgi/fonts/ibmbw.std</code>	<code>./usr/lib/cgi/fonts/ibmco.bld</code>
<code>./usr/lib/cgi/fonts/ibmco.mon</code>	<code>./usr/lib/cgi/fonts/ibmco.std</code>
<code>./usr/lib/cgi/fonts/ibmega.bld</code>	<code>./usr/lib/cgi/fonts/ibmega.mon</code>
<code>./usr/lib/cgi/fonts/ibmega.std</code>	<code>./usr/lib/cgi/fonts/1ja.1</code>
<code>./usr/lib/cgi/fonts/1ja.2</code>	<code>./usr/lib/cgi/fonts/1ja.3</code>
<code>./usr/lib/cgi/fonts/1jb.1</code>	<code>./usr/lib/cgi/fonts/1jb.2</code>

<code>./usr/lib/cgi/fonts/1jb.3</code>	<code>./usr/lib/cgi/fonts/1jb.4</code>
<code>./usr/lib/cgi/fonts/1jb.5</code>	<code>./usr/lib/cgi/fonts/1jb.6</code>
<code>./usr/lib/cgi/fonts/1jf.1</code>	<code>./usr/lib/cgi/fonts/1jf.2</code>
<code>./usr/lib/cgi/fonts/1jf.3</code>	<code>./usr/lib/cgi/fonts/1jf.4</code>
<code>./usr/lib/cgi/fonts/1jf.5</code>	<code>./usr/lib/cgi/fonts/1jf.6</code>
<code>./usr/lib/cgi/fonts/1j1.1</code>	<code>./usr/lib/cgi/fonts/1j1.2</code>
<code>./usr/lib/cgi/fonts/1j1.3</code>	<code>./usr/lib/cgi/fonts/1jm.1</code>
<code>./usr/lib/cgi/fonts/1jm.2</code>	<code>./usr/lib/cgi/fonts/1jm.3</code>
<code>./usr/lib/cgi/fonts/1jn.1</code>	<code>./usr/lib/cgi/fonts/1jn.2</code>
<code>./usr/lib/cgi/fonts/1jn.3</code>	<code>./usr/lib/cgi/fonts/thinkjet.bld</code>
<code>./usr/lib/cgi/fonts/thinkjet.std</code>	<code>./usr/lib/cgi/fonts/instfont</code>
<code>./usr/sys/io/cn286.o</code>	<code>./usr/sys/io/cn386.o</code>
<code>./usr/sys/mdep/machdep286.0</code>	<code>./usr/sys/mdep/machdep386.0</code>

6.3 Packages in This Set

The SCO CGI Development System consists of the following packages:

<i>SCO Computer Graphics Interface Packages</i>		
NAME	PACKAGE	SIZE IN BLOCKS
ALL	SCOCGI Extended Package	5304
286	SCOCGI 286 Libraries	720
386	SCOCGI 386 Libraries	240
DRIVER	SCOCGI Drivers	3830
TEST	SCOCGI Test Files	70
EGA	SCOCGI EGA Driver	346

7. Programmer's Guide Notes

7.1 Signals

By default, SCO CGI will remove shared memory segments and semaphores for the application when a signal occurs. If the

application wishes to do its own signal handling, it is very important that Close Workstation is called before the application exits. If Close Workstation is not called, SCO CGI will not remove the shared memory segments and semaphores.

8. Setting Environmental Variables

To use SCO CGI, you must export certain environment variables to the XENIX Operating System. This can be done from the operating system, in either the `sh` shell or the `cs`h shell. The environment variables can also be set in `.profile` (`sh`) or `.login` (`cs`h) files which are executed each time a user logs on.

1. Set the `VDIPATH` parameter. This parameter provides a path to the directory in which the device driver files reside. If placed by the installation procedure, this directory is `/usr/lib/cgi`.

Set `VDIPATH` from the `sh` shell by typing:

```
VDIPATH=/usr/lib/cgi
export VDIPATH
```

Alternately, from the `cs`h shell, type:

```
setenv VDIPATH /usr/lib/cgi
```

2. Any device-logical names which will be referenced by the workstation identifier, `work_in` [11] to `work_in` [18], of the Open Workstation routine must be assigned to the appropriate device driver file names. These logical names are user-selectable; the SCO CGI has no pre-assigned logical names. The device driver files must be located in the directory specified by `VDIPATH`.

If you pass the letters 'D','P','S','P','L','A','Y' in *work_in* [11] to *work_in* [18] assign the logical device name from the *sh* shell by typing:

```
DISPLAY=driver_name
export DISPLAY
```

Alternately, from the *esh* shell, type:

```
setenv DISPLAY driver_name
```

3. The device driver file names referenced in step 2 must be assigned to the system's physical devices. (If a device is not assigned, *STDIO* is assumed. See step 5.) There are two methods for doing this.

The first method is to add the assignment of the physical device to the workstation identifier assignment shown in the previous step.

For example, from the *sh* shell, type:

```
DISPLAY='driver_name /dev/tty $n$ n'
export DISPLAY
```

Alternately, from the *esh* shell, type:

```
setenv DISPLAY 'driver_name /dev/tty $n$ n'
```

In general, this is the preferred method, as it reduces the size of the environment.

The second method is to assign the physical device via the driver name.

For example, from the **sh** shell, type:

```
DISPLAY=driver_name
driver_name=/dev/ttynn
export DISPLAY driver_name
```

Alternately, from the **cs**h shell, type:

```
setenv DISPLAY driver_name
setenv driver_name /dev/ttynn
```

4. If you select a printer, you may also want to pipe the output through the system's spooler.

As an example of this, from the **sh** shell, enter the following in place of step 3:

```
printer_driver'|spooler_program'
export printer_driver
```

Alternately, from the **cs**h shell, enter:

```
setenv printer_driver 'spooler_program'
```

where *printer_driver* is the printer device driver name, and *spooler_program* is the name of the program for sending data to the print spooler.

Note

Output redirection is not possible with the *cgabw*, *cgaco*, and *ega* drivers.

5. If you want to receive input from one device and send output to another, you can use the redirection symbols **<** and **>** when assigning physical devices.

As an example of this, enter the following from the `sh` shell in place of step 3:

```
DISPLAY='driver_name </dev/tty< /dev/ttym'  
export DISPLAY
```

Alternately, from the `cs` shell, enter:

```
setenv DISPLAY 'driver_name </dev/tty< /dev/ttym'
```

6. If you want to access the device driver's standard I/O streams instead of a physical device, you can assign the keywords `STDIN`, `STDOUT`, `STDERR` and `STDIO`.

As an example of this, from the `sh` shell, enter the following in place of step 3:

```
DISPLAY='driver_name <STDIN> STDERR'  
export DISPLAY
```

Alternately, from the `cs` shell, enter:

```
setenv DISPLAY 'driver_name <STDIN> STDERR'
```

7. Using redirection described in step 4, you can assign a physical device along with a standard I/O stream.

As an example of this, enter the following from the `sh` shell in place of step 3:

```
DISPLAY='driver_name </dev/tty< /dev/ttynn > STDOUT'  
export DISPLAY
```

Alternately, from the `cs` shell, enter:

```
setenv DISPLAY 'driver_name </dev/tty< /dev/ttynn > STDOUT'
```

Note

The keyword **STDIO** is synonymous with **<STDIN>STDOUT**.

8. If you want to access a regular file instead of a physical device, you can assign the file name. File names assigned without a full directory path-specification will be accessed or created in the device driver process's current working directory. The redirection symbol **>** can be used to truncate the assigned file before sending output. The redirection symbol **>>** can be used to append output to the end of the assigned file.

As an example of this, in place of step 3, enter the following from the sh shell:

```
DISPLAY='driver_name <STDIN>> file_name'  
export DISPLAY
```

Alternately, from the csh shell, enter:

```
setenv DISPLAY 'driver_name <STDIN>> file_name'
```

9. If you send output to a metafile (device driver file **cgmdd**), you may want to assign the output metafile file-name to be used instead of the default, **metafile.dat**.

The commands to do this from the sh shell are:

```
META_OUTPUT=file_name  
export META_OUTPUT
```

From the csh shell, the command is:

```
setenv META_OUTPUT file_name
```

10. Should you desire to send plotter prompts to a display other than the current controlling terminal (which is required for the GRAFTERM driver, `grftmdd`, if the display is currently open), you can set a special environment parameter called MESSAGEPORT. The MESSAGEPORT parameter may also be used to select a display device for output from the metafile Message routine.

The default assignment for this parameter is the device driver's standard I/O streams `stdin` and `stdout`.

The commands to set MESSAGEPORT from the `sh` shell are:

```
MESSAGEPORT=/dev/ttynn
export MESSAGEPORT
```

From the `esh` shell, the command is:

```
setenv MESSAGEPORT /dev/ttynn
```

where *device_name* is the device to which messages are to be routed.

11. It is possible to tune the performance of the drivers by changing the size of the shared memory buffer used by the SCO CGI library to communicate to the drivers. This is done by setting the value of the SHMMAX environment variable. The maximum size allowable is 32 Kbytes; this is the default. It may be set to anything between 2 and 32 Kbytes.

From the `sh` shell, the commands to do this (for a 10 Kbyte buffer) are:

```
SHMMAX=10
export SHMMAX
```

From the `esh` shell, the command is:

```
setenv SHMMAX 10
```

12. To use the raster fonts provided in this release of SCO CGI, it is necessary to set up your FONTS environment variable.

From the Bourne shell (**sh**), the commands to set **FONTS** are:

```
FONTS=directory_path
export FONT
```

From the C-shell (**csh**), the command is:

```
setenv FONT directory_path
```

If you are creating your own fonts, using fonts not in this distribution, or if you have moved fonts out of */usr/lib/cgi/fonts*, then you must run the **instfont** program, */usr/lib/cgi/fonts/instfont*, to set up the file *fontlist.dat* which describes the fonts available. The **instfont** utility requires the **FONT** environment variable to be set to the directory containing the fonts.

If any new fonts are placed into */usr/lib/cgi/fonts*, **instfont** should be executed.

13. Different EGA's have different amounts of memory. To notify to the EGA driver how much memory is on the card, set the **EGAMEM** environment variable.

```
EGAMEM=0    64K
EGAMEM=1    128K
EGAMEM=2    192K
EGAMEM=3    256K
```

9. Compiling and Running the Test Program

A test program **CGITEST** has been included with SCO CGI. The purpose of this document is to provide operational information about **CGITEST**.

CGITEST is a program that tests various opcodes of a device driver running SCO CGI. The test produces a maximum of twenty separate displays called frames. Frames for which the opcodes are not supported in a particular device will not be generated. For more information, refer to the *SCO CGI Programmer's Guide* and the *SCO CGI Device Driver Supplement*.

9.1 Compiling Procedure

1. Verify that the `VDIPATH` parameter is set correctly. Refer to step number 1 in the previous section.
2. Select a graphics output device. The demonstration program uses the logical device name `DISPLAY`. Verify that the proper environment variables for this device have been set. Refer to step number 2 in the previous section.
3. Copy the demonstration files to your work directory:

```
cp /usr/lib/cgi/sample/cgitest.c .  
cp /usr/lib/cgi/sample/makefile .
```
4. To compile the demonstration program, enter:

```
make
```
5. To execute the demonstration program, enter:

```
cgitest
```


If you receive any error messages or your device does not display any graphics, then:

- a. Check to see that all environment variables have been properly set.
- b. Check to see that file names are correct.
- c. Review the capabilities of your device, listed in the *SCOCGI Device Driver Supplement*.
- d. Compare any error codes with those listed in Appendix A of the *SCOCGI Programmer's Guide*.

9.2 Test Program Frame Descriptions

9.2.1 Frame 1

Frame 1 tests MARKERS, POLYLINES, FILLED AREA and GRAPHIC TEXT:

- First a box, enclosing the entire display surface, is drawn using the default color of 1 and the default line style of 1.
- Above a grid and an arrow inside the box, are two centered graphic text strings using bottom alignment that names the test and company.
- Inside the box on the left side, a column of six defined markers, in one to six colors, of standard (default) size is drawn. The number of colors is limited by the device's maximum number of colors.

- The center contains a 6 by 6 grid that is a series of polylines with each row-column combination in a different color (up to the device maximum). Marker number 6 (diamond) is drawn at each row-column intersection on a diagonal of the intersections from lower-left to upper-right.
- Across the bottom is a row of six VDC (Normalized Device Coordinates) specific size markers centered on a horizontal line. The line and the six markers are drawn in color 1. Actual device limits may produce fewer sizes but the sizes should increase from left to right.
- Above the test name is a horizontal row of six different line style segments drawn with increasing color index values (1-6). Each segment's length is approximately 1/8 of the screen width.
- To the right of the grid is a solid filled area (arrow) in color 1.
- Below the arrow and to the right of the lower-right corner of the grid is the version number of the device driver being used with the test.
- Above the line-styles test is a prompt message in graphic text telling the user to press <Return> to continue.

9.2.2 Frame 2

Note

Frames 2-5 are cursor mode tests. If your device does not support cursor addressable mode, press <Return> for each frame.

Frame 2 tests CURSOR ADDRESSING using the device's escape functions:

- The display surface is cleared and cursor text mode is enabled.
- The test outputs a series of upper-case 'A' characters diagonally downward across the screen from the upper-left corner of the display surface.
- After each 'A' is written, commands are issued to move the cursor down and then right one character position. One 'A' is written for each row on the display so that the lower-right corner of the display on display with the same numbers of rows and columns.
- Every other column is skipped because the cursor advances a column automatically after writing each character; a move right command is issued.
- The user is then prompted to continue.

9.2.3 Frame 3

Frame3 tests REVERSE VIDEO and ERASE TO END OF LINE.

- **The display is not cleared from Frame 2 and a series of uppercase 'B' characters is written in reverse video, beginning in the lower right corner of the display surface.**
- **Each 'B' written is followed by a move left and a move up command which result in a vertical column of 'B' characters at the right edge of the display.**
- **The cursor is then moved to the center of the display and an erase to end of line command is issued. A single 'B' will be erased on the center row of the display.**
- **The user is prompted to continue.**

9.2.4 Frame 4

Frame 4 tests ERASE TO END OF SCREEN:

- **Again the previous frame is not cleared and the cursor is moved down four lines and an erase to end of screen command is issued.**
- **An inquiry is made to the device and the returned cursor position is saved.**
- **The user is prompted to continue.**

9.2.5 Frame 5

Frame 5 tests HOME CURSOR and ERASE TO END OF SCREEN.

- **The previous screen is not cleared until a home cursor command is issued, followed by an erase to end of screen command.**

- An uppercase 'C' character is printed in the home position and the cursor is then moved to the previously saved cursor position (center screen plus four lines down) and another uppercase 'C' is written.
- The user is then prompted to continue.

9.2.6 Frame 6

Frame 6 tests WRITING MODE:

- The cursor text mode is exited and the display surface cleared. Four columns of output are generated by this test frame.
- The left column is the number of the writing mode being tested (1-16, bottom to top).
- The second column is the number of the writing mode written in the selected mode on a normal background.
- The third column is first written with a solid bar in index 1 color, and then the writing mode number selected is written over the bar using the selected mode.
- The rightmost column is the graphic string "ABCabc" written in REPLACE MODE.
- A horizontal line of fixed VDC length (3200) and position then overwrites a portion of the string. The number of characters overwritten with the line is dependent on the device's character size.
- The user is prompted to continue.

9.2.7 Frame 7

Frame 7 tests GENERALIZED DRAWING PRIMITIVES (GDPS).

- The display is cleared and sixteen bars are drawn in two rows across the bottom of the screen.
- The bottom row of bars is filled with the fill area color index incremented from 0 to 7, the fill area interior style index as 0, 1 and the remainder at interior style 3 (hatch), and the fill area style incremented from 0 to 7 (or to the device maximum).

Note

The lower left bar is not visible as it is drawn in background color.

- The second row of bars is filled with the fill area color index incremented from 1 to 9, the fill area interior style index at 2 (pattern), and the fill area style incremented from 6 to 14 (pattern styles 1 to 6 are the same as hatch styles 1 to 6). The fill area color index is set to 2 for the remainder of this frame's tests.
- A 90 degree arc GDP is then drawn on the left side of the display above the bars.
- A 90 degree pie slice with pattern as the fill interior style and a fill style index of narrow spaced 45 degree lines is drawn to the right of the arc.
- Two circles of equal radii and different interiors are drawn to the right of the pie slice.
The first circle is drawn with an interior style set to PATTERN and a fill style index of 1. The second circle (right-most) is drawn with an interior style set to HATCH and a fill area hatch index of 3.
- A 90 degree elliptical arc GDP is then drawn on the left side of the display above the circular arc.

- A 90 degree elliptical pie slice with pattern as the fill interior style and a fill style index of narrow spaced 45 degree lines is drawn to the right of the arc.
- Two ellipses of equal radii and different interiors are drawn to the right of the pie slice.
The first ellipse is drawn with an interior style set to PATTERN and a fill style index of 1. The second ellipse (rightmost) is drawn with an interior style set to HATCH and a fill area hatch index of 3.
- The user is prompted to continue.

9.2.8 Frame 8

Frame 8 tests GRAPHIC TEXT ROTATION:

- The display is cleared and the string "ABCabc" is written and then rotated and written again in 45 degree increments.

Note

On devices which do not support character rotation or support only 90 degree rotations, the string will be rotated to the devices' ability.

- The user is prompted to continue.

9.2.9 Frame 9

Frame 9 tests GRAPHIC TEXT SIZE and POSITION:

- The display is cleared and eleven 'A' characters are drawn on a horizontal line in increasing sizes and color indices from left to right across the center of the screen.

Note

The first 'A' character will not be visible because it is drawn in background color.

The number of different sizes and colors shown may be limited by the device's capabilities.

- A single 'A' character is then drawn on a type 5 polymarker below the line of 'A' characters. The lower left baseline of the character should be at the center of the polymarker.
- The user is then prompted to continue.

9.2.10 Frame 10

Frame 10 tests OUTPUT CELL ARRAY:

- The display is cleared and two cell arrays with the same parameters are drawn, one on top of the other. Each cell array is filled with up to six different colors depending on the device's maximum.

Note

On devices that do not support pixel operation capabilities, the area of the cell array will only be outlined.

- The top cell array is then inquired on and after saving the results, the user is prompted to continue.

9.2.11 Frame 11

Frame 11 tests INQUIRE CELL ARRAY.

- The display is cleared and, if the results of the inquiry of frame 10 resulted in no error and the device has pixel capabilities, the top cell array is redrawn.
- If an error status was returned or the device has no pixel capabilities, nothing is drawn.
- The user is then prompted to continue.

9.2.12 Frame 12

Frame 12 tests LINE WIDTH:

- Horizontal lines of equal length but increasing width (to the device's maximum) will be drawn starting from the bottom of the display. The thinnest lines will be at the bottom and the widest at the top.

Note

On a device that supports only one line width, only one line will appear.

- The user is then prompted to continue.

9.2.13 Frame 13

Frame 13 tests GRAPHIC TEXT FONTS.

- For each font supported by the device hardware, a string of 45 characters is written beginning with a 'space' and ending

with a 'tilde' character.

The string is shown below enclosed in quotes:

```
" !#$%&'()*+,-./0123456789:;<=>?ABC[^\`abc{}`"
```

Note

Devices with only one font will show only one row of characters.

The line can be clipped without displaying the entire line. This can happen on devices with large characters that cause the string to run off the right edge of the display surface.

- The user is then prompted to continue.

9.2.14 Frames 14–16

Frames 14 through 16 test COLOR REPRESENTATION. These three frames display two colored horizontal lines with the color index written to the left of each line:

- Initially the screen is cleared and the color indices 1 and 2 (if the device has > 2 indices) are redefined.
- The two lines and the associated color index numbers are then drawn.
- In all except the last frame, the realized values are then inquired on and the values returned are used to set the color representation for the next frame.

All three frames should therefore appear to be the same color.

- The user is then prompted to continue after each of the frames.

9.2.15 Frame 17

Frame 17 tests REQUEST LOCATOR.

- This frame clears the display and requests the user to input six points. The points are selected by moving a graphic cursor around on the display surface with the graphics input device and then making each selection by terminating the input. Input is terminated by pressing an alpha key if the GIN device is a keyboard or by pressing one of the mouse's buttons if the device is a mouse. For every input function, one of the six available cursors will be used.
- Inking is turned on for the first point and alternately turned off and on for each of the remaining five points.
- Rubberbanding is cycled for the six points as follows:

Off for the first and fourth points.

Rubberband line for the second and fifth points.

Rubberband rectangle for the third and sixth points.

- After the input of the sixth point, the generated polygon is filled.
- The user is then prompted to continue.

9.2.16 Frame 18

Frame 18 tests REQUEST CHOICE.

- The display is cleared and the user is requested to press a function key.
- An inquiry is made to the device and a message is displayed echoing the choice (e.g., "key=4" for function key 4).
- The user is then prompted to continue.

9.2.17 Frame 19

Frame 19 tests REQUEST STRING with echo off:

- The display is cleared and the user is requested to enter a string of characters. No characters are displayed as the characters are typed but the entire string is displayed when the Return key is pressed.

Note

The default line edit characters, 'backspace' (ctl-H) and 'kill line' (ctl-U) are valid inputs.

- The user is then prompted to continue.

9.2.18 Frame 20

Frame 20 tests REQUEST STRING with echo on.

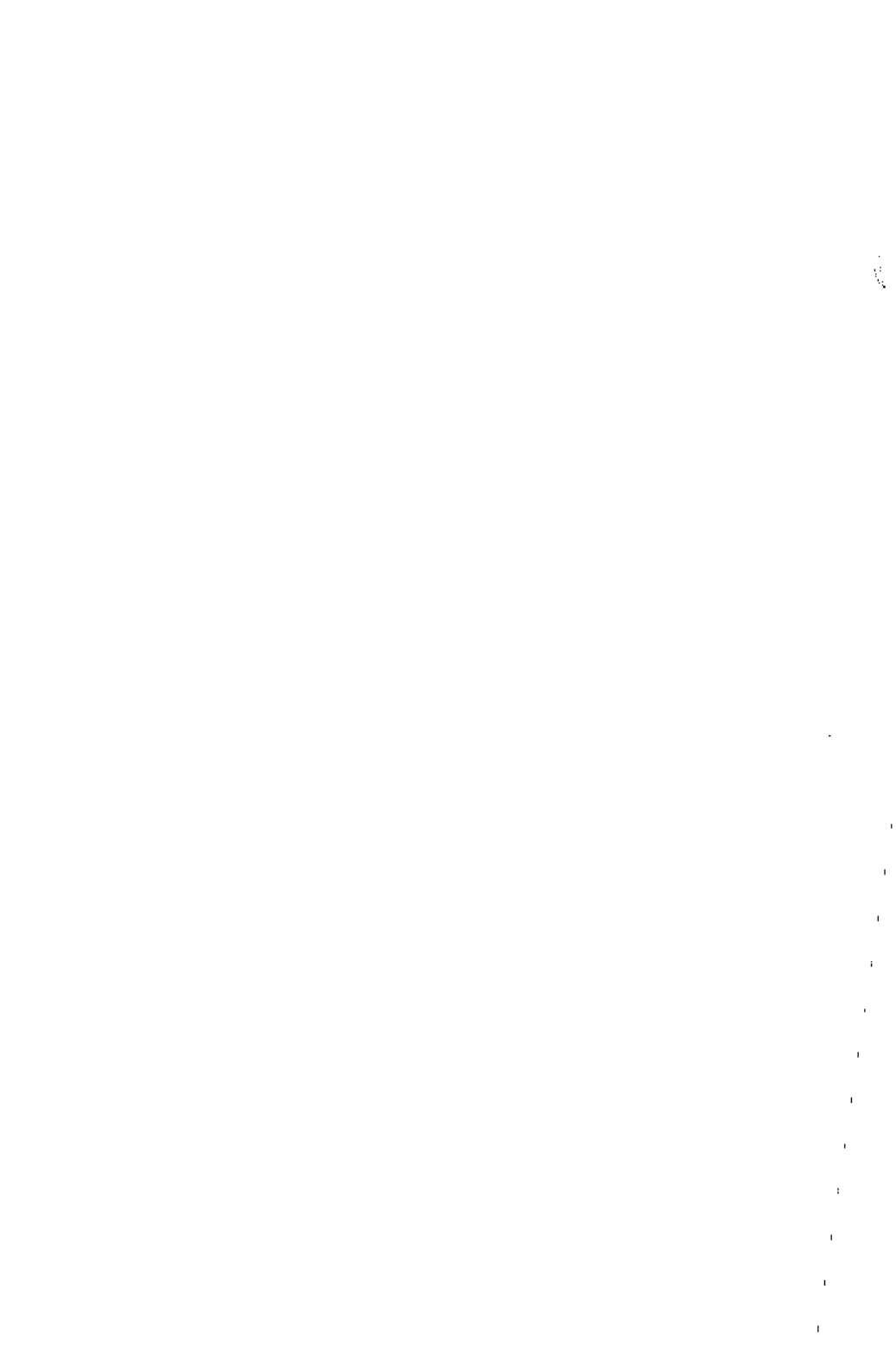
- The display is cleared and the user is requested to enter a string of characters. The characters are echoed to the display as the keys are pressed.
The entire string is echoed again when the Return key is pressed.
- The user is then prompted to continue.

The test clears the screen and returns the user to the operating system.

10. Known Bugs and Omissions in This Release

The following are known bugs and omissions in SCO CGI as of this date:

- Abnormal exits from SCO CGI may leave lockfiles in /tmp. In addition, semaphores and shared memory segments may not be removed when an abnormal exit occurs. See Appendix A of the *SCO CGI Programmer's Guide, Error Codes*, for more information.
- This release of SCO CGI includes the C Language binding only.
- The cgaco, cgabw, and ega drivers only support choice keys 1-10 (F1-F10).
- The SCO CGI drivers may behave erratically when repeated signals are received. To interrupt SCO CGI processes, press the key once.
- The ega driver is only supported in configurations of 64 and 256 Kbytes and is not supported on monochrome monitors.
- When running the EGA driver with a normal color display, the EGA environment variable must be set to HR3 or MR3.
- Due to SCO CGI's multiple process nature, running SCO CGI applications on machines with less than 1 Mbyte of memory may provoke system thrashing.
- This release of SCO CGI is not supported under the XENIX System V Controlled Release 2.2.2 for the PS/2.
- This release of SCO CGI is not supported on the HP Vectra.
- The EGA driver may experience some difficulty when opening multiple raster fonts. To avoid driver problems, open only one raster font at a time.





11-23-87

014-730-012

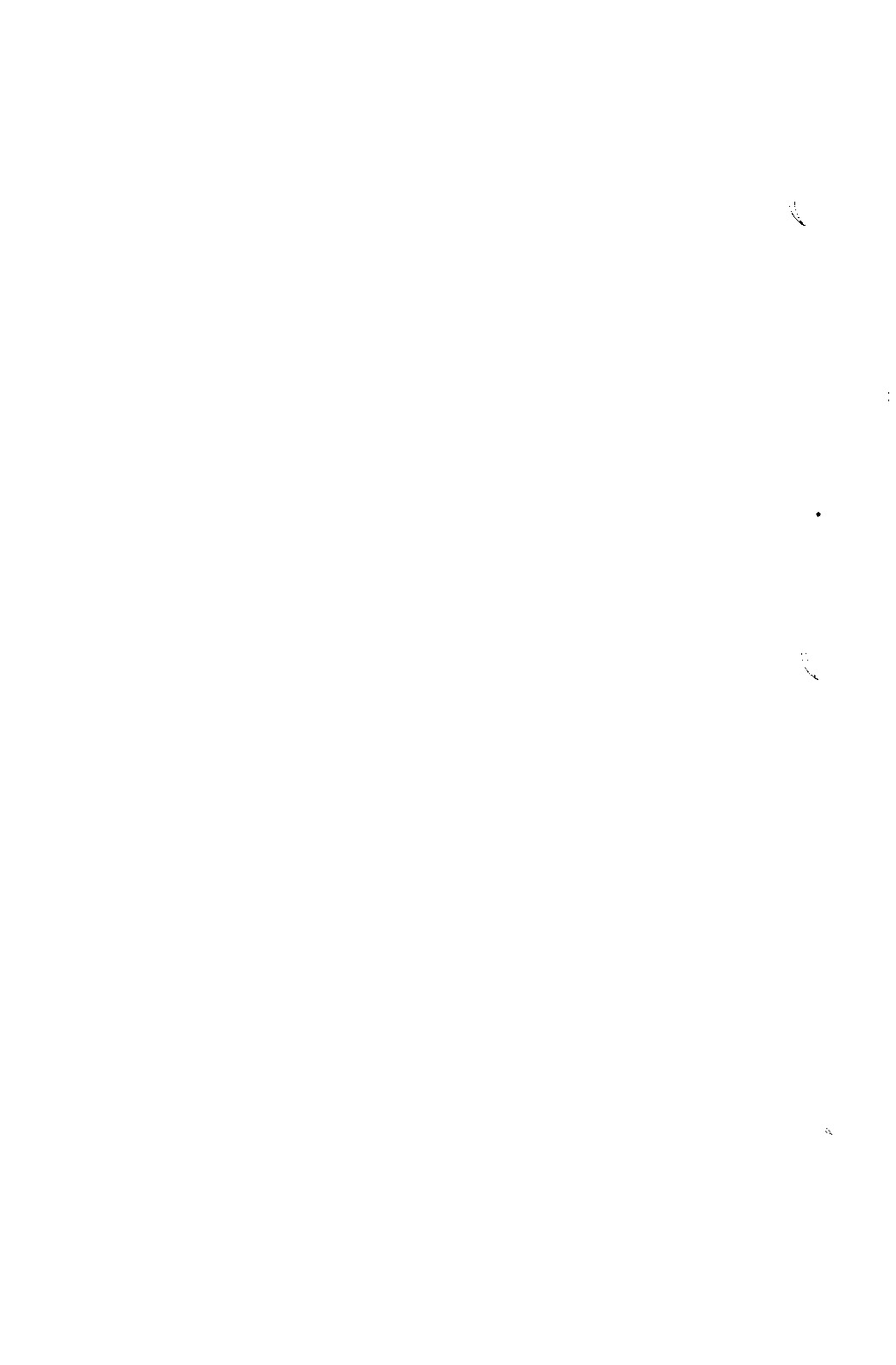
)



SCO XENIX®

Development System

Computer Graphics Interface



A Note to Developers of XENIX Graphics Applications

Thank you for purchasing the SCO XENIX Development System with the SCO Computer Graphics Interface (SCO CGI).

Continuing in our tradition of expanding XENIX environments, SCO has adapted the popular GSS*CGI™ to work closely with SCO XENIX. At this time, the SCO CGI product is included with the SCO XENIX Development System at no extra charge. With the SCO CGI, developers now have the tools to create high-quality graphics applications packages for the XENIX environment.

Along with the SCO CGI package, also at no additional cost, are selected graphics device drivers* that developers can use to test the functionality of their graphics applications on various peripherals. These drivers are not, however, for further distribution without the prior consent of The Santa Cruz Operation, Inc.

SCO offers the SCO Graphics Run Time Package to run graphics applications developed with the SCO CGI. Please call The Santa Cruz Operation, Inc. regarding the availability of the the SCO Graphics Run Time Package, as well as for additional information.

*The graphics device drivers included are:

Apple LaserWriter	HP Plotter
Enhanced Graphics Adapter (EGA)	HP Laserjet
Color Graphics Adapter (CGA)	HP Thinkjet
Epson MX/FX 80/100 printers	GSS Grafstation
GSS Metafile	



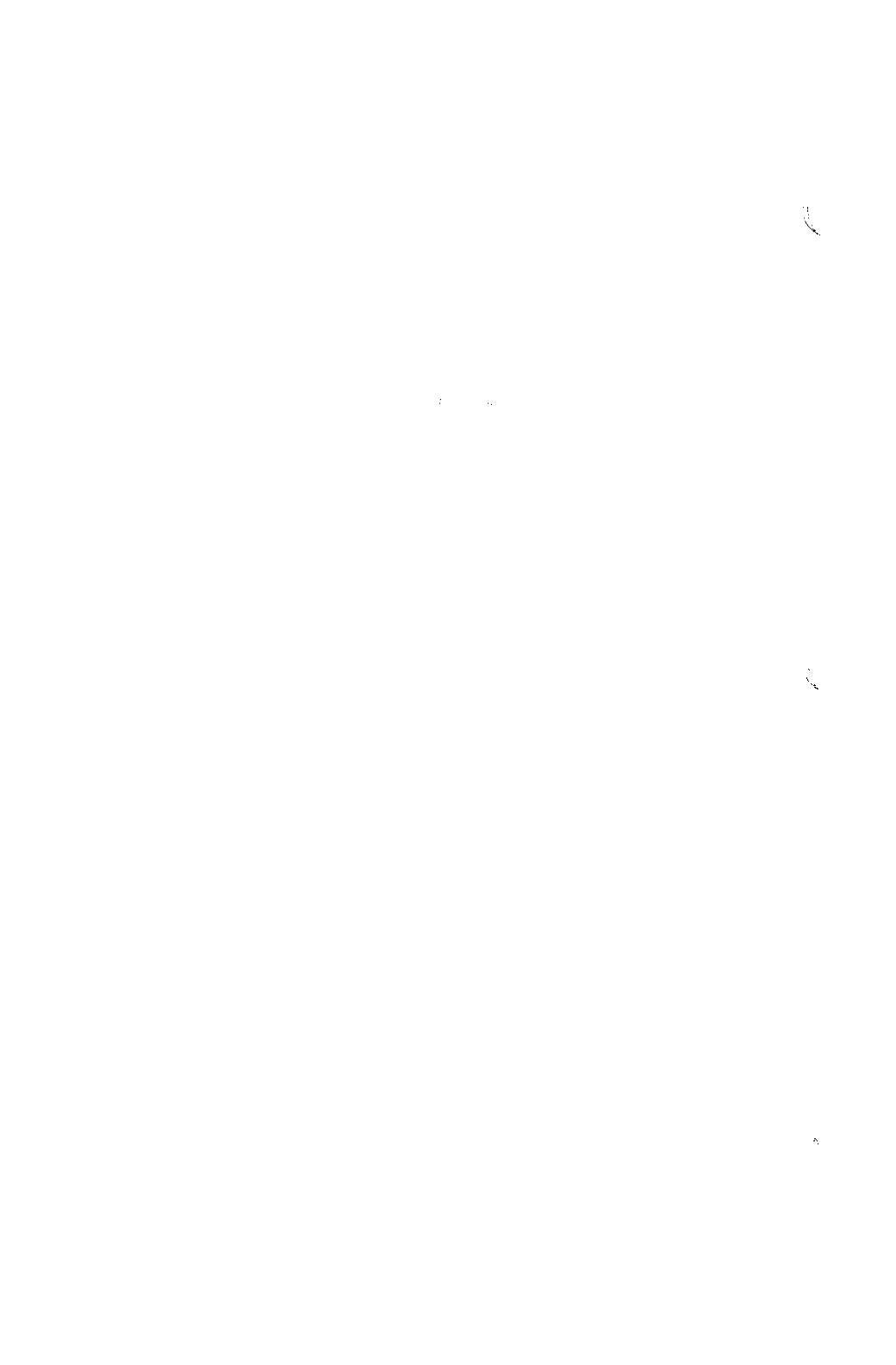
Replace this Page
with Tab Marked:

**Programmer's
Guide**

SCO XENIX[®]

Development System

CGI Programmer's Guide



Information in this document is subject to change without notice and does not represent a commitment on the part of The Santa Cruz Operation, Inc. nor Microsoft Corporation. The software described in this document is furnished under a license agreement or non-disclosure agreement. The software may be used or copied only in accordance with the terms of the agreement. It is against the law to copy this software on magnetic tape, disk, or any other medium for any purpose other than the purchaser's personal use.

Portions © 1986, 1987 Graphic Software Systems, Inc.
All rights reserved.
Portions © 1987 The Santa Cruz Operation, Inc.
All rights reserved.

ALL USE, DUPLICATION, OR DISCLOSURE WHATSOEVER BY THE GOVERNMENT SHALL BE EXPRESSLY SUBJECT TO RESTRICTIONS AS SET FORTH IN SUBDIVISION (b) (3) (ii) FOR RESTRICTED RIGHTS IN COMPUTER SOFTWARE AND SUBDIVISION (b) (2) FOR LIMITED RIGHTS IN TECHNICAL DATA, BOTH AS SET FORTH IN FAR 52.227-7013.

This document was typeset with an IMAGEN® 8/300 Laser Printer.

Microsoft, MS-DOS, and XENIX are registered trademarks of Microsoft Corporation. Microsoft Macro Assembler, Microsoft BASIC, Microsoft FORTRAN, Microsoft Pascal, and Microsoft C are trademarks of Microsoft Corporation. IMAGEN is a registered trademark of IMAGEN Corporation. GSS, GSS-DRIVERS, and the GSS logo are registered trademarks, and GSS*CGI and GSS*GRAFSTATION are trademarks of Graphic Software Systems, Inc. IBM is a registered trademark and IBM Macro Assembler is a trademark of International Business Machines, Inc.

SCO Document Number: XG-6-12-87-1.0
Processed: Wed Jun 10 14:03:12 PDT 1987

Contents

1 Overview

- 1.1 About This Manual 1-1
- 1.2 About SCO CGI 1-2
- 1.3 Using SCO CGI 1-2
- 1.4 The SCO Product Environment 1-4
- 1.5 Creating Graphics Applications 1-4
- 1.6 Graphics Standards and SCO CGI 1-5

2 The Graphics Model

- 2.1 Overview 2-1
- 2.2 The Graphics Model 2-1
- 2.3 Raster Technology and Bitmaps 2-2
- 2.4 CGI Coordinate System 2-3
- 2.5 Program Construction 2-6
- 2.6 Overview Of The SCO CGI Functions 2-9
- 2.7 Error Codes 2-17
- 2.8 Using The SCO CGI Programmer's Guide 2-17

3 SCO CGI Functions

- 3.1 Overview 3-1
- 3.2 Pseudocode 3-1
- 3.3 Master Function List 3-1
- 3.4 Application Data 3-7
- 3.5 Clear Workstation 3-8
- 3.6 Close Workstation 3-9
- 3.7 Copy Bitmap 3-10
- 3.8 Create Bitmap 3-12
- 3.9 Create Cursor 3-14
- 3.10 Cursor Down 3-19
- 3.11 Cursor Home 3-20
- 3.12 Cursor Left 3-21
- 3.13 Cursor Right 3-22
- 3.14 Cursor Up 3-23
- 3.15 Delete Bitmap 3-24
- 3.16 Delete Cursor 3-26
- 3.17 Direct Cursor Address 3-27
- 3.18 Display Graphics Input Cursor 3-29
- 3.19 Enter Cursor Addressing Mode 3-31
- 3.20 Erase To End Of Line 3-33
- 3.21 Erase To End Of Screen 3-34

- 3.22 Escape 3-35
- 3.23 Exit Cursor AddressingMode 3-36
- 3.24 Hard Copy 3-37
- 3.25 Inquire Addressable Character Cells 3-38
- 3.26 Inquire AlphaText Capabilities 3-39
- 3.27 Inquire AlphaText Cell Location 3-42
- 3.28 Inquire AlphaText Font Capability 3-44
- 3.29 Inquire AlphaText Position 3-47
- 3.30 Inquire AlphaText StringLength 3-48
- 3.31 Inquire BackgroundMode 3-49
- 3.32 Inquire Bitmap Formats 3-50
- 3.33 Inquire BytePixel Array 3-51
- 3.34 Inquire CellArray 3-54
- 3.35 Inquire CGIError 3-56
- 3.36 Inquire Clip Rectangle 3-57
- 3.37 Inquire Color Representation 3-59
- 3.38 Inquire Current CursorTextAddress 3-61
- 3.39 Inquire Current Fill Area Attributes 3-62
- 3.40 Inquire Current GraphicsText Attributes 3-64
- 3.41 Inquire Current LineAttributes 3-66
- 3.42 Inquire Current Marker Attributes 3-68
- 3.43 Inquire Cursor Description 3-70
- 3.44 Inquire Deferral Mode 3-72
- 3.45 Inquire Displayable Bitmaps 3-73
- 3.46 Inquire Drawing Bitmap 3-75
- 3.47 Inquire Fill Area Representation 3-77
- 3.48 Inquire Graphics Input Cursor 3-79
- 3.49 Inquire GraphicsText Extent 3-80
- 3.50 Inquire GraphicsText Font Character 3-83
- 3.51 Inquire GraphicsText Font Description 3-85
- 3.52 Inquire GraphicsText Font Metrics 3-88
- 3.53 Inquire GraphicsText Representation 3-90
- 3.54 Inquire Input Extent 3-93
- 3.55 Inquire Integer Pixel Array 3-94
- 3.56 Inquire Line Representation 3-96
- 3.57 Inquire Marker Representation 3-99
- 3.58 Inquire Optimum Pattern Size 3-101
- 3.59 Load CGI 3-103
- 3.60 Message 3-106
- 3.61 Open Workstation 3-108
- 3.62 Output AlphaText 3-116
- 3.63 Output Arc 3-118
- 3.64 Output Bar 3-122
- 3.65 Output BytePixel Array 3-124
- 3.66 Output CellArray 3-127
- 3.67 Output CGIError 3-131
- 3.68 Output Circle 3-132
- 3.69 Output Cursor AddressableText 3-134

3.70 Output Ellipse 3-136
3.71 Output Elliptical Arc 3-138
3.72 Output Elliptical Pie Slice 3-141
3.73 Output Filled Area 3-143
3.74 Output GraphicsText 3-146
3.75 Output Integer Pixel Array 3-148
3.76 Output Pie Slice 3-150
3.77 Output Polyline 3-152
3.78 Output Polymarker 3-155
3.79 ReadCursorKeys 3-158
3.80 Remove CGI 3-161
3.81 Remove Graphics Input Cursor 3-162
3.82 Request Choice 3-163
3.83 Request Locator 3-164
3.84 Request String 3-167
3.85 Request Valuator 3-169
3.86 Reset To Defaults 3-170
3.87 Reverse Video Off 3-172
3.88 Reverse Video On 3-173
3.89 Sample Choice 3-174
3.90 SampleLocator 3-175
3.91 Sample String 3-178
3.92 Sample Valuator 3-180
3.93 Select Drawing Bitmap 3-181
3.94 Select Graphics Input Cursor 3-182
3.95 Set Alpha Text Color Index 3-184
3.96 Set Alpha Text Font And Size 3-186
3.97 Set Alpha Text Line Spacing 3-189
3.98 Set Alpha Text Overstrike Mode 3-191
3.99 Set Alpha Text Pass Through Mode 3-192
3.100 Set Alpha Text Position 3-193
3.101 Set Alpha Text Quality 3-194
3.102 Set Alpha Text Sub/Superscript Mode 3-196
3.103 Set Alpha Text Underline Mode 3-198
3.104 Set Background Color Index 3-199
3.105 Set Background Mode 3-201
3.106 Set Clip Rectangle 3-203
3.107 Set Color Representation 3-205
3.108 Set Color Table 3-207
3.109 Set Cursor Text Attributes 3-208
3.110 Set Cursor Text Color Index 3-211
3.111 Set Deferral Mode 3-212
3.112 Set Fill Area Representation 3-214
3.113 Set Fill Color Index 3-216
3.114 Set Fill Interior Style 3-218
3.115 Set Fill Style Index 3-220
3.116 Set Graphics Text Alignment 3-227
3.117 Set Graphics Text Character Height 3-232

- 3.118 Set Graphics Text Color Index 3-235
- 3.119 Set Graphics Text Font 3-237
- 3.120 Set Graphics Text Representation 3-239
- 3.121 Set Graphics Text String Baseline Rotation 3-241
- 3.122 Set Input Extent 3-243
- 3.123 Set Line Color Index 3-245
- 3.124 Set Line Cross Section 3-247
- 3.125 Set Line Edit Characters 3-249
- 3.126 Set Line Representation 3-250
- 3.127 Set Line Type 3-253
- 3.128 Set Line Width 3-255
- 3.129 Set Marker Color Index 3-257
- 3.130 Set Marker Height 3-259
- 3.131 Set Marker Representation 3-260
- 3.132 Set Marker Type 3-262
- 3.133 Set Pen Speed 3-264
- 3.134 Set User Line Type 3-265
- 3.135 Set Writing Mode 3-267
- 3.136 Update Workstation 3-270

A Error Codes

- A.1 Overview A-1
- A.2 Error Code Descriptions A-1

B SCO CGI Architecture

- B.1 Overview B-1
- B.2 Device Drivers B-1
- B.3 BindingPart B-1

C Font File Architecture

- C.1 Raster Font File Architecture C-1
- C.2 Font File Format C-1

D ASCII Code Chart

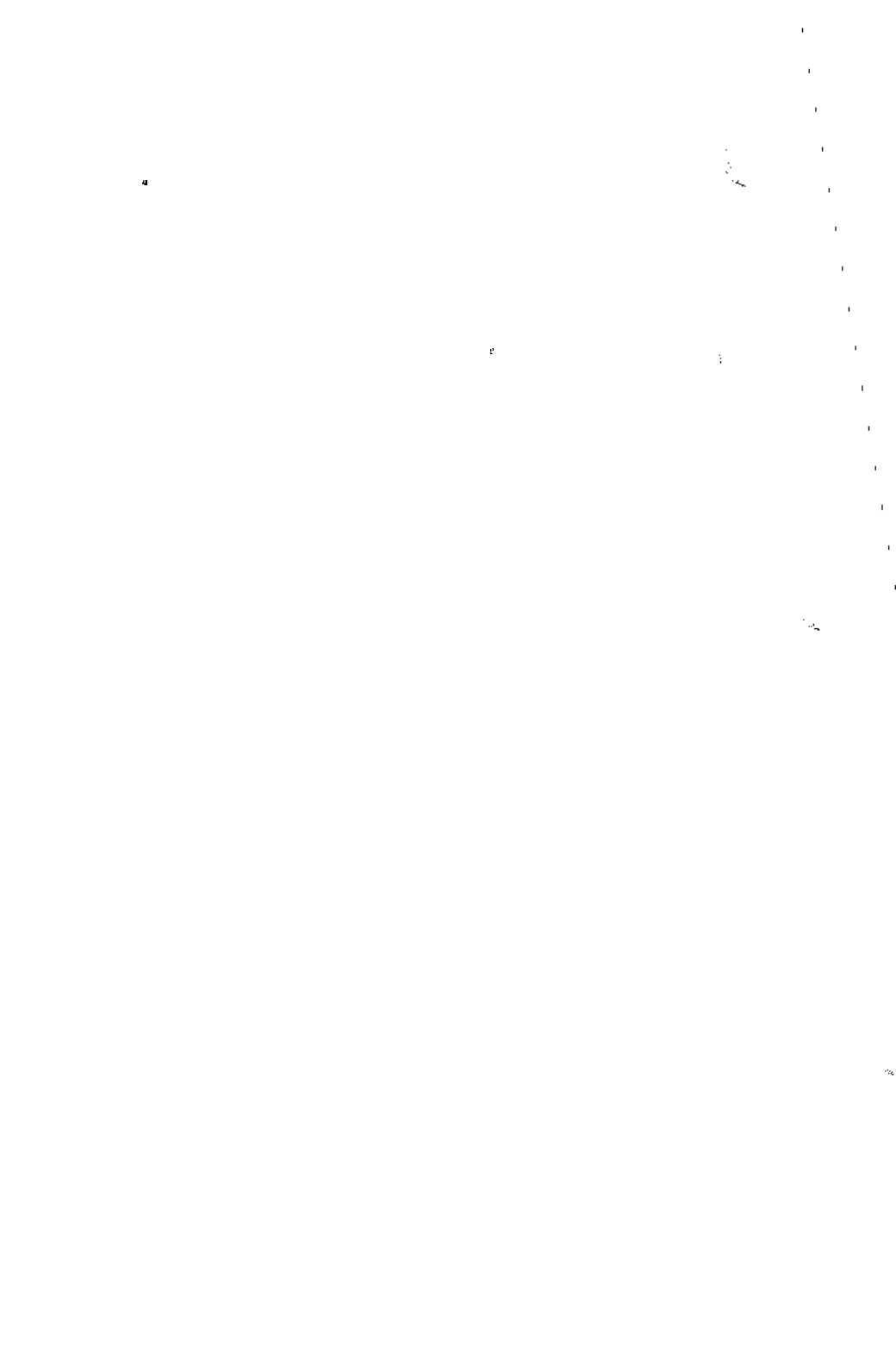
- D.1 Overview D-1

G Glossary

Chapter 1

Overview

- 1.1 About This Manual 1-1
- 1.2 About SCO CGI 1-2
- 1.3 Using SCO CGI 1-2
 - 1.3.1 Coordinate Transformation 1-2
 - 1.3.2 Graphics Output 1-2
 - 1.3.3 Attribute Specification 1-3
 - 1.3.4 Text Manipulation 1-3
 - 1.3.5 Graphics Input 1-3
 - 1.3.6 Status Inquiry 1-3
 - 1.3.7 Metafile Generation 1-3
 - 1.3.8 Bitmap Manipulation 1-3
 - 1.3.9 Pixel Output 1-4
- 1.4 The SCO Product Environment 1-4
- 1.5 Creating Graphics Applications 1-4
 - 1.5.1 Low Level Calls 1-5
 - 1.5.2 High Level Calls 1-5
- 1.6 Graphics Standards and SCO CGI 1-5



1.1 About This Manual

This manual is intended for computer graphics programmers and application developers. It provides the generic information necessary to understand, install and use SCO CGI (SCO Computer Graphics Interface) in the application environment. Specific language syntax for each of the SCO CGI functions is presented in separate reference guides.

The manual is organized as follows.

Chapter 1	Overview describes the manual organization, summarizes the development and implementation of graphics standards, introduces the SCO product line and SCO CGI.
Chapter 2	The Graphics Model describes the computer graphics reference model and gives an overview of the SCO CGI functions.
Chapter 3	SCO CGI Functions contains detailed generic descriptions of each of the CGI functions including parameters and lists of related functions. Chapter 3 also contains a master reference list that categorizes all the CGI functions by class and sub-class.
Appendix A	Error Codes lists all the error codes returned by SCO CGI and an explanation of the cause for each.
Appendix B	SCO CGI Architecture describes the resident and non-resident components of SCO CGI.
Appendix C	Font Files describes font file architecture to enable programmers to create additional fonts.
Appendix D	ASCII Chart
Glossary	
Index	

In addition to this manual, the following SCO CGI publications are also available.

- Programming language specific binding reference guides - accompany the *SCO CGI Programmer's Guide* and provide the language specific syntax for each SCO CGI function.

- *Device Driver Supplement* - provides device specific data.

1.2 About SCO CGI

SCO CGI is a device independent interface to graphics devices. It provides an interface that allows a computer to control several graphics devices simultaneously without regard for their individual characteristics.

SCO CGI provides for the output of graphics primitives (for example: polylines, polymarkers, text, bars, circles, arcs) with control over primitive attributes (for example: color, fill style, line style). It supports output to multiple devices and is capable of performing several forms of graphics input (positional, character string, selection from menus, single value).

SCO CGI version 1.0 provides sophisticated facilities that allow creation of device independent graphics and windowed user interfaces in a bit-mapped environment. It also provides support for features such as mouse-driven pop-up menus, icons, flexible and higher quality text output and definable clipping regions. SCO CGI version 1.0 is upwardly compatible with GSS-DRIVERS versions 1.0 and above.

1.3 Using SCO CGI

When SCO CGI is installed, its presence is generally transparent to the user except that graphics capabilities are available. Device drivers are loaded as needed. See the release notes for more information.

SCO CGI provides the following graphics capabilities.

1.3.1 Coordinate Transformation

The Virtual Device Coordinate (VDC) system allows graphics information to be specified for all devices in an identical way, regardless of the device used. VDC coordinates are automatically transformed to device coordinates by SCO CGI. See Chapter 2, *The Graphics Model*, for more information.

1.3.2 Graphics Output

Graphics primitive functions generate graphic objects on a display surface. The available primitives are:

arcs	elliptical arcs	polylines
bars	elliptical pie slices	polymarkers
circles	grids of cells	text
ellipses	pie slices	

1.5.1 Low Level Calls

Functions can be accessed directly through low level assembly language routine calls with the appropriate parameter lists. However, the calling mechanism is system dependent. The resulting application is device-independent, but operating system-dependent.

1.5.2 High Level Calls

The preferred method is to use language bindings that access graphics functions directly as high level language calls with formal parameters. This ensures computer independence. A separate binding for each programming language provides an interface between the language source and the CGI interface. You simply follow the definitions when making calls in your source code. The binding interface is linked in as an external function library after compilation; ensuring that your application will be completely portable across any SCO CGI compatible system. After linking, you may load and run your graphics application just as you would any other program.

The exact process for generating a new application is system-specific. Detailed information can be found in the release notes.

1.6 Graphics Standards and SCO CGI

SCO CGI is consistent with the Computer Graphics Virtual Device Interface Baseline Document, Revision 3, March, 1985 (X3H3/85-47); developed by the American National Standards Institute (ANSI) and the International Standards Organization (ISO). Our adherence to these evolving standards provides source code portability, device independence, and insurance against product obsolescence.

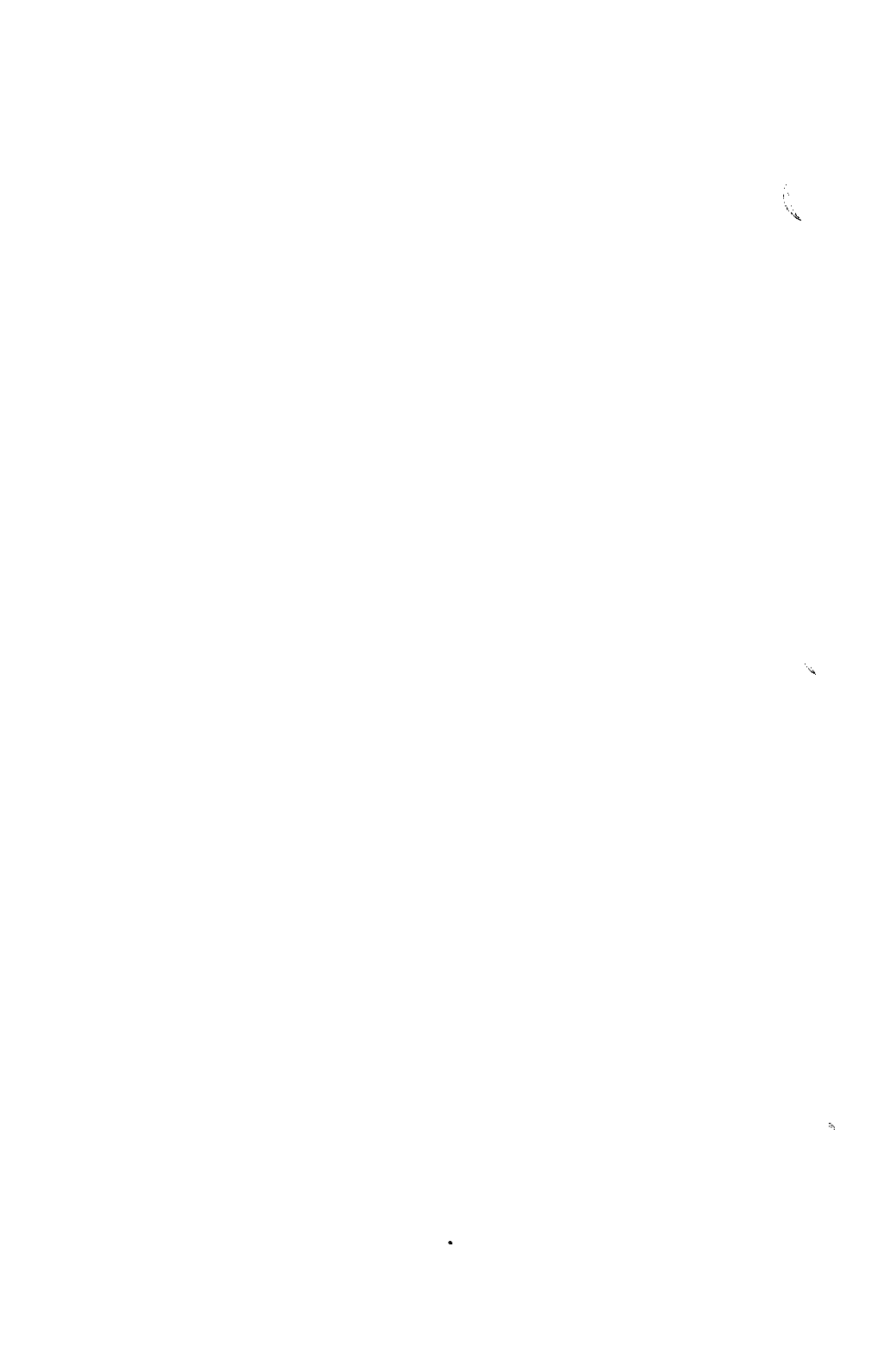
SCO CGI provides device-specific drivers based on the CGI standard for graphics peripherals. In addition to standardizing graphics applications, SCO CGI greatly reduces the programming effort required to create a graphics program. Low level primitive operations and device peculiarities are hidden from the programmer. Interfaces for high level languages such as BASIC, FORTRAN, C and Pascal enable the programmer to incorporate graphics as function calls based on the CGI standard interface.

100

Chapter 2

The Graphics Model

- 2.1 Overview 2-1
- 2.2 The Graphics Model 2-1
 - 2.2.1 Workstations 2-1
 - 2.2.2 Graphics Input 2-1
 - 2.2.3 Transformations 2-2
 - 2.2.4 Graphics Output 2-2
- 2.3 Raster Technology and Bitmaps 2-2
- 2.4 CGI Coordinate System 2-3
 - 2.4.1 Coordinate Transform Modes 2-4
- 2.5 Program Construction 2-6
 - 2.5.1 Using The Open Workstation Function 2-7
 - 2.5.2 Graphics Mode and Cursor Addressing Mode 2-8
- 2.6 Overview Of The SCO CGI Functions 2-9
 - 2.6.1 Control Functions 2-9
 - 2.6.2 Bitmap Functions 2-10
 - 2.6.3 Output Functions 2-10
 - 2.6.4 Attribute Functions 2-15
 - 2.6.5 Input Functions 2-16
 - 2.6.6 Inquiry Functions 2-17
- 2.7 Error Codes 2-17
- 2.8 Using The SCO CGI Programmer's Guide 2-17



2.1 Overview

The purpose of this chapter is to describe the computer graphics reference model. A description of this model is essentially a description of the architecture of SCO CGI. An understanding of the model (and SCO CGI architecture) will help you use SCO CGI in an application program. A detailed description of each function, its arguments and operation is contained in Chapter 3 **SCO CGI Functions**. The language-specific calling sequences for the SCO CGI functions are provided in the language binding reference booklets.

2.2 The Graphics Model

The graphics model grew out of the standards effort discussed in Chapter 1. It provides the programmer with a conceptual framework to aid in organizing an application and using graphics tools, such as SCO CGI. Also, by providing a structural context, the model helps a graphics programmer communicate with other programmers and operators.

The model is built on the following basic concepts:

- workstations
- graphics input
- transformations
- graphics output.

2.2.1 Workstations

The model defines a workstation as zero or one output devices and zero or more input devices such as a keyboard or a mouse. The CGI uses identifiers to refer to a single generic graphics device such as a display, printer or plotter. However, many workstations consist of a display and at least one input device, for example, a CRT with keyboard arrow keys.

2.2.2 Graphics Input

Information input from a device as a result of an action by the operator is called graphics input. For example, graphics input occurs when an operator selects a location on the display surface with a mouse.

2.2.3 Transformations

The point coordinates of graphics images must often be translated between different coordinate systems. This occurs at the application level when an operator scales or rotates an image. It also occurs when coordinates are modified so that an image can be reproduced on graphics display devices of different sizes. The CGI standard is based on a special coordinate system called the Virtual Device Coordinate (VDC) System. In the VDC area, locations are represented as Cartesian coordinates with values between -32768 and 32767 , representing the full coordinate space for all devices. SCO CGI transforms the VDC points into the appropriate device coordinates. See **CGI Coordinate System** later in this chapter for more information on the VDC area.

2.2.4 Graphics Output

Graphics output functions are abstractions of the basic actions an output device can perform, such as drawing a line or displaying text. Graphics primitives result in visible images on the display surface. Graphics attributes modify the appearance of primitives by changing various characteristics, such as color and fill pattern. See **Output Functions** in this chapter for an overview of these functions and Chapter 3, **SCO CGI Functions** for detailed information about the implementation of the Output Functions.

2.3 Raster Technology and Bitmaps

Recent trends in the human interface of computer systems have brought a demand for high performance graphics systems and software. Increasingly, applications are relying on communications with the user through graphics. Graphics programs employ icons, user-defined windows, and pop-up menus. The development of less expensive and more powerful display devices is making all of this possible. The tool utilized is called raster technology and is based on television technology.

In the raster display, the refresh memory is arranged as a two-dimensional grid or array. The entry at a particular point (defined by its row and column) stores the brightness and/or color value of a corresponding point (defined by x and y coordinates) on the screen. This is accomplished through a simple one-to-one relationship. Each screen location and memory location is referenced by an x coordinate and a y coordinate. The top row of a matrix corresponds to the top row (scan line) of the display, and so on. Image refreshing is done by a sequential raster scan through the display buffer by scan line rather than by output primitive.

Because each memory location defines one point element of the image, each screen location and its corresponding buffer location is called a *pixel* or *pel* (picture element). A simple refresh buffer has one bit per pixel and defines a two color (monochrome) image since each pixel can be either on or off. In a typical color displaysubsystem, more bits per pixel are stored to hold information regarding color or grayscale (intensity).

Because the new powerful graphic display sub-systems can store and scan these images very quickly, higher resolution graphics with more flexibility can be achieved. The flexibility is possible with the manipulation of the display buffer array that stores the image. This buffer array is commonly called a *bitmap* because it is a collection of pixels that are represented by one or more bits per pixel. Graphics software, such as SCO CGI, can make excellent use of the bitmap for creating and manipulating high resolution graphics images.

2.4 CGI Coordinate System

The CGI is addressed in Virtual Device Coordinates (VDCs). The VDC area is an abstract space in which graphics functions define virtual images. Theoretically, VDC space is a two-dimensional Cartesian coordinate space of infinite precision and infinite extent. In reality, only a subset of VDC space is realizable. With 16-bit integer coordinates, the VDC range is the set of integers in the range -32768 to 32767. The actual size of the display space is -32K to 32K, the visible region is from 0 to 32K (refer to Figure 2-1).

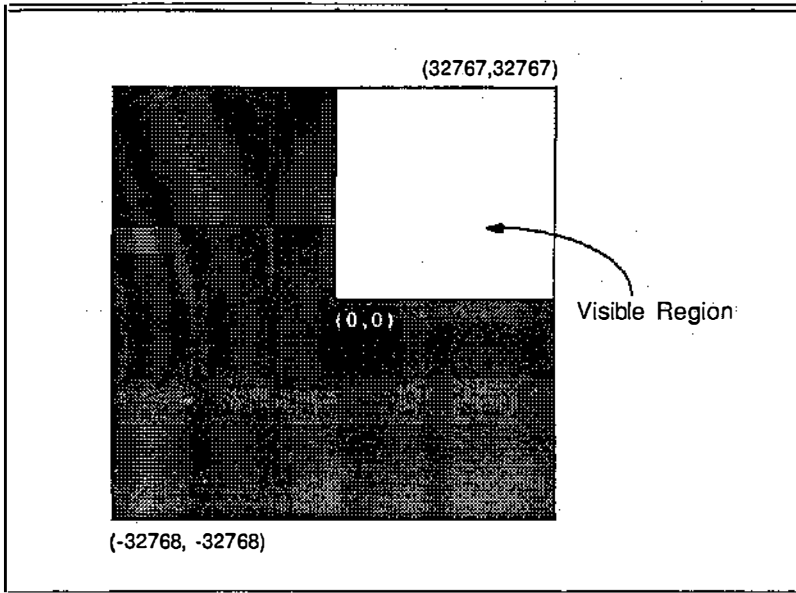


Figure 2-1. *VDC space, showing the visible and non-visible regions*

2.4.1 Coordinate Transform Modes

The VDC extent of the display surface on the opened device is determined by the value of the coordinate transform mode flag when the device is opened. There are four transform modes.

- 0 full screen
- 1 preserve aspect ratio
- 2 device units
- 3 short axis

Mode 0 - fullscreenmode

Mode 0 maps 0 through 32767 in x and y to the full extent of the display surface. It provides the application with a full positive 16-bit integer address space for graphics output. Aspect ratio is not preserved. Therefore, lines of the same VDC length projected along the x and y axes will appear to be different physical lengths on devices that do not have the same number and/or size of pixels on both axes.

Using this mode ensures that all the graphics information will appear on the display surface since all VDC points are displayable. However, distortion will occur if the display device does not map VDC units to equal physical distances in both directions (this happens on devices with non-unity aspect ratios). The result is squares that turn into rectangles.

Mode 1 - preserve aspectratio mode

Mode 1 maps 0 through 32767 to the full extent of the geometrically longer display surface axis only; it maps a subset of VDC space to the shorter axis. The preserve aspect ratio mode provides the application with output with a 1:1 aspect ratio. Lines of the same VDC length projected along the x and y axes will be the same physical length. The physical VDC extent is device-dependent. The VDC extents are given to the application in the *work_out* array when the Open Workstation function is invoked. This mode is useful for applications that require a 1:1 aspect ratio for graphics output.

Mode 0 unburdens the application from doing a specific device-dependent transform. The advantage of Mode 1 is that pictures can be easily transported between devices with the assumption that a unity (square) aspect ratio is used.

Mode 2 - device units mode

In Mode 2 all coordinates are specified in physical device-dependent units. The device units mode enables the application to address physical device-dependent coordinates. The physical VDC extent is device-dependent. The VDC extents are provided to the application in the *work_out* array at open workstation time. This mode is sometimes referred to as the raster address mode. This mode is useful for raster-oriented applications that need to access specific physical pixels.

Mode 3 - short axis mode

Mode 3 maps 0 through 32767 to the full extent of the geometrically shorter display surface axis. It maps 0 through 32767 to a subset of the longer display surface axis in such a way as to preserve a unity aspect ratio. The

short axis mode provides the application with output that provides a 1:1 aspect ratio while showing the entire VDC extent (0 through 32767) on both the x and y axes. This is done by mapping the entire VDC extent (0 through 32767 on both the x and y axis) onto the largest rectangle that can fit on the display device such that a 1:1 aspect ratio is preserved and 0,0 is mapped to the lower left corner of the display device. Thus, space is left unused at the top or right side of the display device if the aspect ratio of the device is not 1:1. Lines of the same VDC length projected along the x and y axes will be the same physical length.

SCO CGI uses information returned from the device driver when the workstation is opened to transform the virtual coordinates into device-specific coordinates that are in device-specific units (raster lines or plotter steps, for example). This frees an application from performing any device-dependent transforms.

2.5 Program Construction

A typical graphics application will consist of the following five steps:

1. Setup graphics control.
2. Define graphics primitive attributes.
3. Output graphics primitives.
4. Input primitives or inquire on status.
5. Terminate graphics.

The intermediate three steps may be repeated numerous times prior to the last step. The general flow of a graphics application program is illustrated in the Figure 2-2.

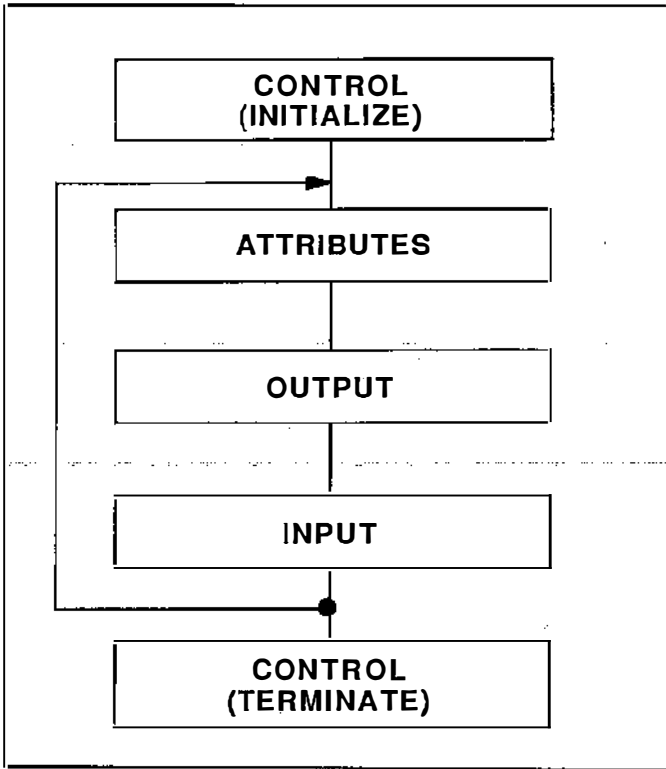


Figure 2-2. *Graphics Program Flow*

2.5.1 Using The Open Workstation Function

The first command the user invokes is *Open Workstation*. This command defines the type of device to be used, loads the device into memory if necessary, and returns information to the user regarding the capabilities of the device.

Coordinate Transformation Flag

One of the key parameters specified by *Open Workstation* is the coordinate transformation flag. This flag indicates the aspect ratio of the display surface. Refer to *CGI Coordinate System* earlier in this chapter for a detailed discussion of the coordinate system and the four coordinate transformation modes.

Prompting Flag

Another parameter that must be provided to the Open Workstation command is the prompting flag. This is only important when the graphics output device requires user attention; for example, when a printer or plotter has run out of paper. The prompting flag indicates whether the user is prompted or whether the device driver ignores the paper change commands. This may be useful for preliminary debugging runs of a program, when the actual number of colors output is not important.

Device-Specific Data

Open Workstation also returns an array of information detailing the capabilities of the requested device, including the number of colors available, the number of text sizes available, and the number of line styles available. The programmer uses this information when developing applications for particular devices. For example, if a device has multiple colors, different lines of data on a chart can be represented with solid lines in different colors. If black and white are the only colors available, the application can use different line styles (combinations of dashes, dots and spaces) to differentiate the lines of data on the same chart output to a different device.

Another important device-specific parameter returned by Open Workstation is the number of different text sizes available. A few devices allow text to be continuously scaled from a few units to 32767 units high. Most other devices provide only four to six discrete text sizes. The programmer should consider the number of scaling options and the actual text sizes needed for the application.

2.5.2 Graphics Mode and Cursor Addressing Mode

After opening the workstation, the programmer should make sure the device is in the proper control mode. The default condition sets Graphics Mode to *on* and Cursor Addressing Mode to *off*. Certain functions work only when Cursor Addressing Mode is on, and other functions only when Cursor Addressing Mode is off. Cursor Addressing Mode is applicable only to CRT devices.

Cursor addressable text is not compatible with graphics text and alpha text. Graphics text and alpha text are compatible with each other and can be displayed simultaneously on the same display surface. The user must remember to call the Enter Cursor Addressing Mode function prior to executing any cursor addressable commands.

If the device is not in Cursor Addressing Mode when a cursor addressing command is executed, the exact results are not well defined. This may hang the device and require reinitialization. In some instances the device may ignore the cursor addressing command. The effect depends on the

state of the device prior to the cursor addressing command. Issuing graphics commands when Cursor Addressing Mode is on produces similar results.

2.6 Overview Of The SCO CGI Functions

The SCO CGI graphics functions can be divided into six groups:

- Control functions
- Bitmap functions
- Output functions
- Attribute functions
- Input functions
- Inquiry functions

2.6.1 Control Functions

The Control Functions allow the application program to control various aspects of the graphics subsystem.

Device-specific functions

For ultimate device control, SCO CGI allows the programmer to set the pen speed on a plotter, set the raster writing mode, generate hard copy, and send device driver specific commands.

Initialization and termination functions

Open Workstation initializes a graphics device and sets defaults for attributes. It also returns information to the caller about the characteristics and capabilities of the device. This must be the first graphics operation performed in a program.

Close Workstation terminates graphics operations to a device. This must be the last graphics operation performed in a program.

Metafile CGI functions

The Message function allows message text that is not part of a picture to be placed in the metafile. This text is passed on to the metafile interpreter and displayed for the operator at interpretation time. It is intended to permit the display of special device-dependent information needed to process a Computer Graphics Metafile (CGM). There is no control over the position or appearance of the message text.

The Application Data function allows an application program to store and access private data in a metafile. When retrieved, this data is not processed in anyway by the metafile interpreter but is available to the application.

These functions are valid only when the CGM driver is loaded.

Workstation control functions

Clear Workstation clears the surface of the workstation. It clears a CRT screen, prompts for new paper on a plotter, displays all pending graphics to a printer, and advances to top-of-form.

Update Workstation displays all pending graphics on the output device or consolidates all pending input on the input device, such as a scanner.

2.6.2 Bitmap Functions

The Bitmap Functions provide the ability to deal with raster images. These functions allow you to create, copy, select, and delete bitmaps. The programmer can use the Bitmap Functions to develop applications that make use of pop-up menus, windows, and icons.


2.6.3 Output Functions

The Output Functions generate graphics objects (primitives) on the display surface. Point locations are specified by giving x and y coordinates in the VDC space.

Arc

This function draws an arc that is defined by a center point, the radius, and the starting and ending angles.

Bar



This function draws a bar (rectangular area) that is defined by two diagonally opposite vertices. The interior may be filled with a pattern.


CellArray

This function draws a rectangular grid of colored rectangles of a specified size at a specified position. The colors that make up the array are defined by a set of color indices.

Circle

This function draws a circle that is defined by a center point and the radius. The interior may be filled with a pattern.

Ellipse



This function draws an ellipse with the center point at x,y and the specified x radius and y radius. The interior may be filled with a pattern.


Elliptical Arc

This function draws an elliptical arc with its center point at x,y and the specified x radius, y radius, start and end angles. The start and end angles are skewed to the same degree that the ellipse is a skewed circle.

Elliptical Pie Slice

This function draws an elliptical pie slice with its centerpoint at x, y and the specified x radius, y radius, start and end angles. The start and end angles are skewed to the same degree that the ellipse is a skewed circle. The interior may be filled with a pattern.

Filled Area



This function causes an area bounded by a specified set of vertices to be filled with a pattern.

Pie Slice

This function draws a pie slice that is defined by its center, radius, starting angle, and ending angle. The interior may be filled with a pattern.

Polyline

This function draws a single vector or a series of connected vectors specified by their vertices. A polyline must have at least a beginning and an end point specified.

Polymarker

This function draws a marker symbol at each specified location.

Pixel Array

The pixel array is a device-independent means of placing and retrieving blocks of color information. The color information in the pixel array maps directly to the pixels of the device. The starting point and color information are specified in a device-independent manner, but the appearance of the final image depends directly on the resolution and aspect ratio of the output device. An $m*n$ array of device-independent color indices is mapped to an $m*n$ array of device-dependent pixels.

Text

SCO CGI allows flexibility when creating and manipulating text. Two control modes are provided for text manipulation: graphics mode and cursor addressing mode. There are three different kinds of text: alpha text, graphics text, and cursor text. Text manipulations possible with all three kinds of text are listed in Table 2-1.

Alpha text displays a text string at a specified location. Variable interline spacing, underlining capabilities and other attributes used for generating formal documents are available for alpha text. Alpha text mode enables the programmer to tailor text to specific requirements, particularly in mixing word processing with graphics. To control alpha text, you must be in the graphics mode.

Graphics text also displays a text string at a specified location. However, graphics text characters can be rotated and scaled and are available in multiple fonts. Graphics mode allows text to be controlled like other graphics primitives. To control graphics text, you must be in the graphics mode.

Cursor text displays a text string on a fixed rectangular grid. It is used with CRTs, particularly in the generation of screen menus and forms. Cursor text is available in only one size and font. It is not rotatable and cannot be combined with graphics text, alpha text or with graphics. Cursor text is usually positioned on a discrete, device-dependent grid of rows and columns.

To control the cursor for cursor text you must be in the cursor addressing mode. Cursor addressing mode gives you access to functions used for positioning the standard CRT cursor and placing text on the screen. Cursor addressing is defined on a character cell grid of rows and columns. Rows are the number of lines of text on the screen, and columns are the number of character cells per line. The upper left-hand corner of the screen is row 1, column 1. A typical screen format is 24 rows by 80 columns.

Note

Cursor addressing applies only to CRT devices.

Table 2-1. Text Attributes

MANIPULATIONS	CURSOR	ALPHA	GRAPHICS
Control Mode	Cursor Addressing Mode	Graphics Mode	Graphics Mode
Positioning	character cell boundary	anywhere on display surface	anywhere on display surface
Scaling	No	Yes*	Yes
Rotation	No	No	Yes
Multiple Fonts	No	Yes	Yes
Bold Text	Yes	Yes	Yes
Color Selection	Yes	Yes	Yes
Underlining	Yes	Yes	Yes†
Overstriking	No	Yes	No
Super- and Subscripting	No	Yes	Yes
Quality Levels	No	Yes	No
Line Spacing	No	Yes	Yes
Reverse Video	Yes	Yes**	Yes‡
Blink Text	Yes	No	No
Text Alignment	No	No	Yes
Variable Text Height	No	Yes*	Yes

* The size of alpha text can be changed via the Set Alpha Text Font And Size function.

† Graphics text can be underlined by specifying polyline primitives underneath the character string.

** Reverse video can be attained for alpha text by using the Set Alpha Text Color and Set Background Color functions.

‡ Reverse video can be selected for graphics text with the Set Writing Mode function.

2.6.4 Attribute Functions

The attribute functions control geometric and non-geometric characteristics of output primitives. Geometric characteristics affect the shape or size of a primitive, for example, character height. Geometric attributes are expressed in VDC units. Non-geometric characteristics modify the appearance of primitives without changing their shape, for example, color.

Current attribute values can be queried and changed any time after SCO CGI is initiated.

Character Attributes

There are three kinds of character attributes: cursor text, alpha text, and graphic text. See Table 2-1 for a complete list of the text options.

Polyline Attributes

The polyline attributes control type, width, color, and cross sections.

Polymarker Attributes

The polymarker attributes control type, size, and color.

Fill Attributes

The fill attributes control type, style, and color.

Color Representation

The color representation attribute assigns color values specified in red, green and blue primaries to the color index numbers. Color attributes are then selected by index number.

Background Index

The background color attribute sets a color index for the display background.

Clipping Functions

Objects are specified in a virtual coordinate system in the CGI and must be converted into the appropriate coordinates of the physical device. If the visible space is smaller than the virtual coordinate space, CGI must make those portions of the object outside the visible space invisible. In SCO CGI this is accomplished with the Set Clip Rectangle function. With the Set Clip Rectangle function you can specify which portion of the virtual coordinate space is viewable. If the coordinates outside the visible space were not clipped, the image displayed would vary from device to device. Refer to **CGI Coordinate System** earlier in this chapter for more discussion of the virtual coordinate space.

2.6.5 Input Functions

Input functions return information from the operator.

Locator

The Locator Function returns the point location of the graphics input device (for example, a mouse, joystick, or trackball) in VDC units.

Valuator

The Valuator Function returns a scalar value between 0 and 32767, corresponding to the status of a valuator device (for example, a potentiometer or slide control).

Choice

The Choice Function returns the status of a choice device (for example, a switch or function key) as an integer between 0 and 32767.

String

The String Function allows text input from the keyboard.

Read Cursor Movement Keys

The Read Cursor Movement Keys Function returns the direction of cursor movement as well as acknowledgement of any special keys pressed.

Sample and Request Modes

The input functions operate in either sample or request mode. In sample mode, the input value is examined and returned immediately. In request mode, the input device is activated and waits for the user to terminate the input process with a device-specific action (the trigger). Then the input value measure is returned. For example, a sample locator input returns the current location of the graphics cursor immediately. In request mode, the location of the graphics cursor is returned when the user terminates the action by pressing an alphanumeric key or a mouse button.

2.6.6 Inquiry Functions

Inquiry functions return information about the current state of the graphics subsystem, including device capabilities and current attributes.

2.7 Error Codes

The value that is returned when a function is invoked informs the user of the success and outcome of the function. The value is derived according to the following general rules.

- A return greater than or equal to zero indicates that no error occurred.
- A negative return always denotes an error. The actual error can be determined by calling `Inquire CGI Error`. The error message that corresponds to the actual error number can be displayed by calling `Output CGIError`.

Refer to Appendix A for complete descriptions of the error codes.

2.8 Using TheSCO CGI Programmer's Guide

The next chapter of this manual is the programmer's reference guide to the SCO implementation of the CGI. It contains detailed descriptions of all the SCO CGI functions, listed in alphabetical order. By using this manual as a desktop reference and referring to Chapter 3 as needed, the programmer will have a complete picture of the Computer Graphics Interface and how best to use it.

Refer to the language specific reference booklet for your programming language for the calling sequences you will need to use SCO CGI in your programming language.



Chapter 3

SCO CGI Functions

- 3.1 Overview 3-1
- 3.2 Pseudocode 3-1
- 3.3 Master Function List 3-1
 - 3.3.1 Control Functions 3-2
 - 3.3.2 Bitmap Functions 3-3
 - 3.3.3 Output Functions 3-3
 - 3.3.4 Attribute Functions 3-3
 - 3.3.5 Input Functions 3-4
 - 3.3.6 Inquiry Functions 3-5
- 3.4 Application Data 3-7
- 3.5 Clear Workstation 3-8
- 3.6 Close Workstation 3-9
- 3.7 Copy Bitmap 3-10
- 3.8 Create Bitmap 3-12
- 3.9 Create Cursor 3-14
- 3.10 Cursor Down 3-19
- 3.11 Cursor Home 3-20
- 3.12 Cursor Left 3-21
- 3.13 Cursor Right 3-22
- 3.14 Cursor Up 3-23
- 3.15 Delete Bitmap 3-24
- 3.16 Delete Cursor 3-26
- 3.17 Direct Cursor Address 3-27

- 3.18 Display Graphics Input Cursor 3-29
- 3.19 Enter Cursor Addressing Mode 3-31
- 3.20 Erase To End Of Line 3-33
- 3.21 Erase To End Of Screen 3-34
- 3.22 Escape 3-35
- 3.23 Exit Cursor Addressing Mode 3-36
- 3.24 Hard Copy 3-37
- 3.25 Inquire Addressable Character Cells 3-38
- 3.26 Inquire Alpha Text Capabilities 3-39
- 3.27 Inquire Alpha Text Cell Location 3-42
- 3.28 Inquire Alpha Text Font Capability 3-44
- 3.29 Inquire Alpha Text Position 3-47
- 3.30 Inquire Alpha Text String Length 3-48
- 3.31 Inquire Background Mode 3-49
- 3.32 Inquire Bitmap Formats 3-50
- 3.33 Inquire Byte Pixel Array 3-51
- 3.34 Inquire Cell Array 3-54
- 3.35 Inquire CGI Error 3-56
- 3.36 Inquire Clip Rectangle 3-57
- 3.37 Inquire Color Representation 3-59
- 3.38 Inquire Current Cursor Text Address 3-61
- 3.39 Inquire Current Fill Area Attributes 3-62
- 3.40 Inquire Current Graphics Text Attributes 3-64
- 3.41 Inquire Current Line Attributes 3-66
- 3.42 Inquire Current Marker Attributes 3-68

- 3.43 Inquire Cursor Description 3-70
- 3.44 Inquire deferral mode 3-72
- 3.45 Inquire Displayable Bitmaps 3-73
- 3.46 Inquire Drawing Bitmap 3-75
- 3.47 Inquire Fill Area Representation 3-77
- 3.48 Inquire Graphics Input Cursor 3-79
- 3.49 Inquire Graphics Text Extent 3-80
- 3.50 Inquire Graphics Text Font Character 3-83
- 3.51 Inquire Graphics Text Font Description 3-85
- 3.52 Inquire Graphics Text Font Metrics 3-88
- 3.53 Inquire Graphics Text Representation 3-90
- 3.54 Inquire Input Extent 3-93
- 3.55 Inquire Integer Pixel Array 3-94
- 3.56 Inquire Line Representation 3-96
- 3.57 Inquire Marker Representation 3-99
- 3.58 Inquire Optimum Pattern Size 3-101
- 3.59 Load CGI 3-103
- 3.60 Message 3-106
- 3.61 Open Workstation 3-108
- 3.62 Output Alpha Text 3-116
- 3.63 Output Arc 3-118
- 3.64 Output Bar 3-122
- 3.65 Output Byte Pixel Array 3-124
- 3.66 Output Cell Array 3-127

- 3.67 OutputCGIError 3-131
- 3.68 Output Circle 3-132
- 3.69 Output CursorAddressableText 3-134
- 3.70 Output Ellipse 3-136
- 3.71 Output Elliptical Arc 3-138
- 3.72 Output Elliptical Pie Slice 3-141
- 3.73 Output Filled Area 3-143
- 3.74 Output GraphicsText 3-146
- 3.75 Output Integer Pixel Array 3-148
- 3.76 Output PieSlice 3-150
- 3.77 Output Polyline 3-152
- 3.78 Output Polymarker 3-155
- 3.79 ReadCursorKeys 3-158
- 3.80 Remove CGI 3-161
- 3.81 Remove Graphics Input Cursor 3-162
- 3.82 Request Choice 3-163
- 3.83 RequestLocator 3-164
- 3.84 Request String 3-167
- 3.85 Request Valuator 3-169
- 3.86 Reset To Defaults 3-170
- 3.87 Reverse Video Off 3-172
- 3.88 Reverse Video On 3-173
- 3.89 Sample Choice 3-174
- 3.90 Sample Locator 3-175
- 3.91 Sample String 3-178

- 3.92 SampleValuator 3-180
- 3.93 SelectDrawingBitmap 3-181
- 3.94 Select Graphics Input Cursor 3-182
- 3.95 SetAlphaText ColorIndex 3-184
- 3.96 Set Alpha TextFont And Size 3-186
- 3.97 SetAlpha Text Line Spacing 3-189
- 3.98 Set Alpha Text Overstrike Mode 3-191
- 3.99 SetAlphaTextPass Through Mode 3-192
- 3.100 Set Alpha Text Position 3-193
- 3.101 Set AlphaTextQuality 3-194
- 3.102 Set Alpha Text Sub/Superscript Mode 3-196
- 3.103 Set Alpha Text Underline Mode 3-198
- 3.104 Set Background ColorIndex 3-199
- 3.105 SetBackgroundMode 3-201
- 3.106 Set ClipRectangle 3-203
- 3.107 Set ColorRepresentation 3-205
- 3.108 Set Color Table 3-207
- 3.109 Set CursorTextAttributes 3-208
- 3.110 Set CursorText Color Index 3-211
- 3.111 SetDeferral Mode 3-212
- 3.112 Set Fill Area Representation 3-214
- 3.113 Set Fill ColorIndex 3-216
- 3.114 Set Fill Interior Style 3-218
- 3.115 Set Fill Style Index 3-220

- 3.116 Set GraphicsTextAlignment 3-227
- 3.117 Set GraphicsText CharacterHeight 3-232
- 3.118 Set GraphicsText ColorIndex 3-235
- 3.119 Set GraphicsTextFont 3-237
- 3.120 Set GraphicsText Representation 3-239
- 3.121 Set GraphicsText StringBaselineRotation 3-241
- 3.122 Set Input Extent 3-243
- 3.123 Set Line ColorIndex 3-245
- 3.124 Set Line Cross Section 3-247
- 3.125 Set Line Edit Characters 3-249
- 3.126 Set Line Representation 3-250
- 3.127 Set LineType 3-253
- 3.128 Set LineWidth 3-255
- 3.129 Set Marker ColorIndex 3-257
- 3.130 Set MarkerHeight 3-259
- 3.131 Set Marker Representation 3-260
- 3.132 Set MarkerType 3-262
- 3.133 Set Pen Speed 3-264
- 3.134 Set UserLineType 3-265
- 3.135 Set WritingMode 3-267
- 3.136 Update Workstation 3-270

3.1 Overview

This section presents the entire set of SCO CGI functions in alphabetical order. Each entry describes the function parameters in terms of their inputs, outputs, and related functions. A brief synopsis and usage description follows each entry.

For specific language parameter requirements and syntax descriptions, please refer to the appropriate language binding reference guide.

The Master Function List that follows contains a logical organization of all the functions into their respective classes.

3.2 Pseudocode

Code examples in this chapter are written in structured English pseudocode so that the purpose of each example and the general organization of the program will be easily recognized.

3.3 Master Function List

The following list categorizes each CGI function by class and sub-class for easy referencing. The six classes are Control, Bitmap, Output, Attribute, Input, and Inquiry. All the functions can be found in this chapter, ordered alphabetically by their generic function name on the pages listed below. Language-specific function names can be found in the binding guides.

3.3.1 Control Functions

Alpha Text Control	Set Alpha Text Position
CGIControl	Load CGI RemoveCGI
CursorText Control	Cursor Down Cursor Home Cursor Left Cursor Right Cursor Up Direct Cursor Address Enter Cursor AddressingMode Erase to End of Line Erase to End of Screen Exit Cursor Addressing Mode
Device Control	Escape HardCopy Set Background Mode Set Deferral Mode Set Pen Speed Set WritingMode
Graphics Cursor Control	Create Cursor Delete Cursor Display Graphics Input Cursor Remove Graphics Input Cursor Select Graphics Input Cursor
Metafile	Application Data Message
Workstation Control	Clear Workstation Close Workstation Open Workstation Reset To Defaults Update Workstation

3.3.2 Bitnap Functions

Copy Bitmap
 Create Bitmap
 Delete Bitmap
 Select Drawing Bitmap

3.3.3 Output Functions

Output Alpha Text
 Output Arc
 Output Bar
 Output Byte Pixel Array
 Output Cell Array
 Output CGI Error
 Output Circle
 Output Cursor Addressable Text
 Output Ellipse
 Output Elliptical Arc
 Output Elliptical Pie Slice
 Output Filled Area
 Output Graphics Text
 Output Integer Pixel Array
 Output Pie Slice
 Output Polyline
 Output Polymarker

3.3.4 Attribute Functions

Alpha Text	Set Alpha Text Color Index Set Alpha Text Font And Size Set Alpha Text Line Spacing Set Alpha Text Overstrike Mode Set Alpha Text Pass Through Mode Set Alpha Text Quality Set Alpha Text Sub/Superscript Mode Set Alpha Text Underline Mode
Background	Set Background Color Index
Clipping	Set Clip Rectangle

SCO CGI Programmer's Guide

Color	Set Color Representation Set Color Table
CursorText	Reverse Video Off Reverse Video On Set Cursor Text Attributes Set Cursor Text Color Index
Fill	Set Fill Area Representation Set Fill Color Index Set Fill Interior Style Set Fill Style Index
GraphicsText	Set Graphics Text Alignment Set Graphics Text Character Height Set Graphics Text Color Index Set Graphics Text Font Set Graphics Text Representation Set Graphics Text String Baseline Rotation
Input	Set Line Edit Characters
Line	Set Line Color Index Set Line Cross Section Set Line Representation Set Line Type Set Line Width Set UserLine Type
Marker	Set Marker Color Index Set Marker Height Set Marker Representation Set Marker Type

3.3.5 Input Functions

Choice	Request Choice Sample Choice
Cursor	Read Cursor Keys
Input Extent	Set Input Extent
Locator	Request Locator Sample Locator

String	Request String Sample String
Valuator	Request Valuator Sample Valuator

3.3.6 Inquiry Functions

Bitmap	Inquire Bitmap Formats Inquire Byte Pixel Array Inquire Displayable Bitmaps Inquire Drawing Bitmap Inquire Integer Pixel Array
Clipping	Inquire Clip Rectangle
Cursor	Inquire Cursor Description Inquire Graphics Input Cursor
Device Capabilities	Inquire Addressable Character Cells Inquire Deferral Mode
Errors	Inquire CGIError
Input Extent	Inquire Input Extent
Primitive	Inquire Background Mode Inquire Cell Array Inquire Color Representation Inquire Current Fill Area Attributes Inquire Current Line Attributes Inquire Current Marker Attributes Inquire Fill Area Representation Inquire Line Representation Inquire Marker Representation Inquire Optimum Pattern Size

Text

Inquire Alpha Text Capabilities
Inquire Alpha Text Cell Location
Inquire Alpha Text Font Capability
Inquire Alpha Text Position
Inquire Alpha Text String Length
Inquire Current Cursor Text Address
Inquire Current Graphics Text Attributes
Inquire Graphics Text Extent
Inquire Graphics Text Font Character
Inquire Graphics Text Font Description
Inquire Graphics Text Font Metrics
Inquire Graphics Text Representation

3.4 Application Data

Class

Control Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>routine</code>	text string indicating user-defined title
<code>data_cnt</code>	number of integers of application data
<code>app_data</code>	name of array containing application data

Outputs

None

Value Returned

0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Message

Description

The Application Data function allows the metafile generator to place application specific data into the metafile. The function name is a user-defined title for whatever the application data element represents.

Note

This function is valid only for the Computer Graphics Metafile driver.

3.5 Clear Workstation

Class

Control Functions

Inputs

dev_handle	CGI handle for a currently open workstation
------------	---------------------------------------------

Outputs

None

Value Returned

0	no error
-1	error occurred; actual error can be retrieved by invoking Inquire CGIError

Related Functions

Close Workstation
Inquire Drawing Bitmap
Open Workstation
Select Drawing Bitmap
Set Background Color Index
Update Workstation

Description

The Clear Workstation function clears the currently selected drawing bitmap to the background color on CRTs, prompts for new paper on plotters, or displays all pending graphics, advances to top-of-form, and clears the subsequent page to the background color on printers.

Invoking the Clear Workstation function is frequently preceded by an input function when using a CRT so that output will not be cleared from the screen before the user has a chance to view the image. This may produce a prompt on some devices. Prompts can be controlled by Open Workstation.

3.6 Close Workstation**Class**

Control Functions

Inputs

dev_handle	CGIhandle for a currently open workstation
------------	--------------------------------------------

Outputs

None

Value Returned

≥0	no error
-1	error occurred; actual error can be retrieved by invoking Inquire CGIError

Related Functions

Clear Workstation
 Open Workstation
 Update Workstation

Description

The Close Workstation function displays any pending graphics, advances to top-of-form on the printer, then stops all graphics output to the specified workstation. It must be called to terminate a program and should be the last graphics function called.

Close Workstation also deletes all user-defined bitmaps and cursors.

3.7 Copy Bitmap

Class

Bitmap Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>bitmap_handle</code>	handle of source bitmap
<code>source_rectangle[0]</code>	<i>x</i> coordinate, in VDC units, of the first end point of a diagonal that describes the source bitmap rectangle
<code>source_rectangle[1]</code>	<i>y</i> coordinate, in VDC units, of the first end point of a diagonal that describes the source bitmap rectangle
<code>source_rectangle[2]</code>	<i>x</i> coordinate, in VDC units, of the second end point of a diagonal that describes the source bitmap rectangle
<code>source_rectangle[3]</code>	<i>y</i> coordinate, in VDC units, of the second end point of a diagonal that describes the source bitmap rectangle
<code>dest_orig[0]</code>	<i>x</i> coordinate, in VDC units, of the lower left corner of the destination rectangle in the currently selected drawing bitmap
<code>dest_orig[1]</code>	<i>y</i> coordinate, in VDC units, of the lower left corner of the destination rectangle in the currently selected drawing bitmap

Outputs

None

Value Returned

≥ 0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Create Bitmap
 Inquire Background Mode
 Inquire Clip Rectangle
 Select Drawing Bitmap
 Set Background Mode
 Set Clip Rectangle
 Set Writing Mode

Description

The Copy Bitmap function copies a rectangular subregion of the source bitmap specified by *bitmap_handle* to the destination bitmap (specified with the Select Drawing Bitmap function), using the current writing mode and background mode.

The rectangular region of the source bitmap is defined explicitly with the *source_rectangle* parameter. The *source_rectangle* parameter is two points representing the diagonal corners of a rectangle. The lower left corner of the destination rectangle is defined explicitly with the *dest_orig* parameter. The upper right corner of the destination rectangle is determined by the width and height of the *source_rectangle* parameter.

If the *source_rectangle* and the destination rectangle are both in the currently selected output bitmap and they overlap, the result of the copy operation will be as if *source_rectangle* had been copied into a temporary area with a writing mode of REPLACE and then copied into the destination bitmap with the current writing mode.

The actual *source_rectangle* is the intersection of the specified *source_rectangle* and the source bitmap extents. The actual destination rectangle is the intersection of the specified destination rectangle, clip rectangle, and destination bitmap extents.

3.8 Create Bitmap

Class

Bitmap Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>xy[0]</code>	<i>x</i> coordinate, in VDC units, of the first end point of a diagonal that describes the bitmap extent
<code>xy[1]</code>	<i>y</i> coordinate, in VDC units, of the first end point of a diagonal that describes the bitmap extent
<code>xy[2]</code>	<i>x</i> coordinate, in VDC units, of the second end point of a diagonal that describes the bitmap extent
<code>xy[3]</code>	<i>y</i> coordinate, in VDC units, of the second end point of a diagonal that describes the bitmap extent
<code>type</code>	0 full depth bitmap (same depth as display) 1 monochrome bitmap (mapped)

Outputs

`bitmap_handle` handle of newly created bitmap

Value Returned

≥ 0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Delete Bitmap
Inquire Displayable Bitmaps
Inquire Drawing Bitmap
Select Drawing Bitmap

Description

The Create Bitmap function creates a virtual bitmap containing the extent. If the creation is successful, its bitmap handle is returned in *bitmap_handle*.

The created bitmap is not automatically selected. The application must invoke the Select Drawing Bitmap function to select the created bitmap.

Physical device bitmaps are special bitmap cases in that they can be actually viewed by the user. The first n bitmaps, where n is returned by the Inquire Displayable Bitmaps function, are called physical device bitmaps (video buffers), and can be made visible to the user. They always exist, and their bitmap handles are 0 through $n-1$. Bitmaps created by the Create Bitmap function are never directly visible to the user, and are given handles greater than or equal to n .

Graphics drawn to the created bitmap that go outside the bitmap extents will be clipped.

The parameter *type* must be set to 0 (full depth) at this time. Monochrome (mapped) bitmaps will be available in a future release of SCO CGI.

Bitmaps are currently supported only on display devices.

3.9 Create Cursor

Class

Control Functions

Inputs

<code>dev_handle</code>	CGIhandle for a currently open workstation
<code>data_bitmap</code>	bitmap handle of the bitmap to be used as the XOR (data) bitmap for the cursor definition
<code>mask_bitmap</code>	bitmap handle of the bitmap to be used as the NAND (mask) bitmap for the cursor definition
<code>x</code>	<code>x</code> coordinate, in VDC units, of the reference point (hot spot) in the data and mask bitmaps; this is the point that will be aligned at the selected input coordinate
<code>y</code>	<code>y</code> coordinate, in VDC units, of the reference point (hot spot) in the data and mask bitmaps; this is the point that will be aligned at the selected input coordinate

Outputs

<code>cursor_handle</code>	the <i>cursor_handle</i> (or index) that will be used when referring to this cursor definition
----------------------------	------------------------------------------------------------------------------------------------

Value Returned

≥ 0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Create Bitmap
 Delete Bitmap
 Delete Cursor
 Inquire Cursor Description
 Inquire Graphics Input Cursor
 Inquire Optimum Pattern Size
 Remove Graphics Input Cursor
 Request Locator
 Select Graphics Input Cursor

Description

The cursor, when displayed on the screen, appears as a rectangular area where the display of each pixel of the rectangle is determined by the corresponding pixels in the data and mask bitmaps. The rectangular extent of the cursor is the smallest rectangle that encloses both the data and mask bitmap extents defined when the bitmap was created with the Create Bitmap function.

The following three operations take place when the cursor is displayed on the screen:

1. The screen image within the cursor extent is temporarily saved.
2. The NOTed mask bitmap is copied using an AND operation to the screen.
3. The data bitmap is copied using an XOR operation to the screen.

Mathematically the operations are:

$$\text{result} = \text{data XOR} (\text{screen AND NOT}(\text{mask}))$$

Although the following discussion explains the functionality in terms of bitmaps that are only one bit deep, the functionality is generalized and applies to full depth bitmaps.

The logical result of the mask and data bitmap operations are defined as follows:

mask	data	Resulting pixel on screen
0	0	same as pixel under cursor
0	1	inverse pixel under cursor
1	0	always set pixel off
1	1	always set pixel on

SCO CGI Programmer's Guide

If all the mask pixels are off (0), the cursor is completely transparent, and the image under the cursor is still visible. Pixels under the black (pixels off) portion of the cursor appear unchanged; pixels under the light (pixels on) portion will be reversed.

If the data or mask bitmaps have operations performed on them after they have been created by the Create Cursor function, the results are device-dependent.

For optimal performance the data or mask bitmaps should be integer multiples of values returned by the Inquire Optimum Pattern Size function.

As an example of this function's use, the following pseudocode segment defines a crosshair graphics cursor (similar to the default graphics input cursor) and then selects the newly defined cursor as the graphics cursor.

```
int cursor_handle
int dev_handle
int data_map
int mask_map
int xy(4)
int valid_width(2)
int valid_height(2)

static char cross_data/      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                           0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                           0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                           0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,
                           0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,
                           0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,
                           0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,
                           0,0,1,1,1,1,1,1,1,1,1,1,1,0,0,0,
                           0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,
                           0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,
                           0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,
                           0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,
                           0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,
                           0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,
                           0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                           0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0/
```

```
static char cross_mask/
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,
0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,
0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,
0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,
0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,
0,1,1,1,1,1,1,1,1,1,1,1,1,0,0,
0,1,1,1,1,1,1,1,1,1,1,1,1,0,0,
0,1,1,1,1,1,1,1,1,1,1,1,1,0,0,
0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,
0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,
0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,
0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,
0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,
0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0/
```

```
{
  Create data bitmap for the cursor, select the newly created bitmap
  for output and then define the data bitmap with Byte Pixel array
}
CREATE_BITMAP(dev_handle, xy, 1, data_map)
SELECT_DRAWING_BITMAP(dev_handle, data_map)
BYTE_PIXEL_ARRAY(dev_handle, origin, 16, -16, valid_width,
  valid_height, cross_data);
{
  Create mask bitmap for the cursor, select the newly created bitmap
  for output and then define the mask bitmap with Byte Pixel array
}
CREATE_BITMAP(dev_handle, xy, 1, mask_map)
SELECT_DRAWING_BITMAP(dev_handle, mask_map)
BYTE_PIXEL_ARRAY(dev_handle, origin, 16, -16, valid_width,
  valid_height, cross_mask);
```

```
{  
  Create the cursor definition with the hotspot set to 7,7 in the  
  bitmap and select the new cursor definition  
}  
CREATE_CURSOR (dev_handle, data_map, mask_map, 7, 7,  
  cursor_handle)  
SELECT_GRAPHIC_INPUT_CURSOR (dev_handle, cursor_handle)  
.  
.  
.
```

3.10 CursorDown**Class**

Control Functions

Inputs

dev_handle	CGIhandle for a currently open workstation
------------	--------------------------------------------

Outputs

None

Value Returned

≥ 0	no error
-1	error occurred; actual error can be retrieved by invoking Inquire CGIError

Related Functions

CursorHome
 CursorLeft
 CursorRight
 CursorUp
 EnterCursorAddressingMode

Description

When you are in Cursor Addressing Mode, the Cursor Down function moves the cursor down one row without altering its horizontal position. If the cursor is already at the bottom margin, the screen will scroll up one line.

Note

This function is applicable only to CRT devices.

3.11 CursorHome

Class

Control Functions

Inputs

dev_handle CGI handle for a currently open workstation

Outputs

None

Value Returned

≥0 no error

-1 error retrieved by invoking Inquire CGIError

Related Functions

CursorDown
CursorLeft
CursorRight
CursorUp
Enter Cursor Addressing Mode

Description

When you are in Cursor Addressing Mode, the Cursor Home function moves the cursor to the upper left corner of the screen (home position).

Note

This function is applicable only to CRT devices.

3.12 CursorLeft**Class**

Control Functions

Inputs

dev_handle	CGIhandle for a currently open workstation
------------	--------------------------------------------

Outputs

None

Value Returned

≥ 0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Cursor Down
 Cursor Home
 Cursor Right
 Cursor Up
 Enter Cursor AddressingMode

Description

When you are in Cursor Addressing Mode, the Cursor Left function moves the cursor left one column without altering its vertical position. No action occurs if the cursor is already at the left margin.

Note

This function is applicable only to CRT devices.

3.13 Cursor Right

Class

Control Functions

Inputs

dev_handle CGIhandle for a currently open workstation

Outputs

None

Value Returned

≥0 no error

-1 error occurred; actual error can be retrieved by
invoking Inquire CGIError

Related Functions

Cursor Down
Cursor Home
Cursor Left
Cursor Up
Enter Cursor Addressing Mode

Description

When you are in Cursor Addressing Mode, the Cursor Right function moves the cursor right one column without altering its vertical position. No action occurs if the cursor is already at the right margin.

Note

This function is applicable only to CRT devices.

3.14 CursorUp**Class**

Control Functions

Inputs

dev_handle	CGIhandle for a currently open workstation
------------	--------------------------------------------

Outputs

None

Value Returned

≥ 0	no error
-1	error occurred; actual error can be retrieved by invoking Inquire CGIError

Related Functions

Cursor Down
 Cursor Home
 Cursor Left
 Cursor Right
 EnterCursor AddressingMode

Description

When you are in Cursor AddressingMode, the Cursor Up function moves the cursor up one row without altering its horizontal position. No action occurs if the cursor is already at the top margin.

Note

This function is applicable only to CRT devices.

3.15 Delete Bitmap

Class

Bitmap Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>bitmap_handle</code>	handle of bitmap to delete

Outputs

None

Value Returned

≥ 0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Close Workstation
Create Bitmap
Delete Cursor
Inquire Drawing Bitmap
Select Drawing Bitmap
Set Fill Area Representation
Set Line Cross Section
Set Marker Representation

Description

The Delete Bitmap function deletes the bitmap specified by *bitmap_handle*. Future references to *bitmap_handle* on this device are invalid.

It is an error to attempt to delete physical bitmaps or bitmaps that are currently being used to describe graphical output primitives. Bitmaps that may not be deleted are:

- physical bitmaps;
- currently selected bitmap;
- bitmap being used for Fill Area Representation;
- bitmap being used as part of a Cursor Description;
- bitmap being used for Line Cross Section;
- bitmap being used for Marker Representation.

3.16 Delete Cursor

Class

Control Functions

Inputs

dev_handle	CGI handle for a currently open workstation
cursor_handle	handle of cursor description to be deleted

Outputs

None

Value Returned

≥0	no error
-1	error occurred; actual error can be retrieved by invoking Inquire CGIError

Related Functions

- Close Workstation
- Create Bitmap
- Create Cursor
- Delete Bitmap
- Inquire Graphics Input Cursor
- Select Graphics Input Cursor

Description

The Delete Cursor function deletes the cursor definition specified by *cursor_handle* previously returned by a call to Create Cursor or Inquire Graphics Input Cursor.

It is an error to attempt to delete a predefined cursor or the currently selected cursor.

This function does not delete the mask and data bitmaps associated with the cursor.

3.17 Direct Cursor Address**Class**

Control Functions

Inputs

dev_handle	CGIhandle for a currently open workstation
row	rownumber (1 to thenumberof rows)
column	column number (1 to the number of columns)

Outputs

None

Value Returned

≥0	no error
-1	error occurred; actual error can be retrieved by invoking Inquire CGIError

Related Functions

Cursor Down
 Cursor Home
 Cursor Left
 Cursor Right
 Cursor Up
 Enter Cursor Addressing Mode
 Inquire Current Cursor Text Address
 Output Cursor A ddressable Text

Description

When you are in Cursor Addressing Mode, the Direct Cursor Address function moves the cursor to the specified position. The position is specified in cursor space where row 1 column 1 is the top left corner of the screen. If you specify a position off the screen, the cursor will not move.

Note

This function is applicable only to CRT devices.

3.18 Display Graphics Input Cursor

Class

Control Functions

Inputs

<code>dev_handle</code>	CGIhandle for a currently open workstation
<code>x</code>	<code>x</code> coordinate of location to place cursor in VDC units
<code>y</code>	<code>y</code> coordinate of location to place cursor in VDC units

Outputs

None

Value Returned

≥ 0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Open Workstation
 Remove Graphics Input Cursor
 Request Locator
 Sample Locator

Description

If you are not in Cursor Addressing Mode, the Display Graphics Input Cursor function displays a graphics input cursor centered on the hot spot. The graphics cursor is the same as the cursor used for feedback by the Request Locator function (for example, a crosshair or arrow). The Request Locator function will automatically display a cursor when needed, therefore you do not need to use this function during a Request Locator invocation.

This function can be used in conjunction with the Sample Locator function to track an input device.

See the Open Workstation parameter *work_out*[45] to determine if the device represented by *dev_handle* is a CRT.

Note

This function is applicable only to CRT devices.

The Display Graphics Input Cursor function does not work if the device is not in Graphics Mode.

3.19 Enter Cursor Addressing Mode**Class**

Control Functions

Inputs

dev_handle	CGI handle for a currently open workstation
------------	---------------------------------------------

Outputs

None

Value Returned

≥ 0	no error
-1	error occurred; actual error can be retrieved by invoking Inquire CGI Error

Related Functions

Cursor Down
 Cursor Home
 Cursor Left
 Cursor Right
 Cursor Up
 Direct Cursor Address
 Erase To End Of Line
 Erase To End Of Screen
 Exit Cursor Addressing Mode
 Inquire Addressable Character Cells
 Inquire Current Cursor Text Address
 Output Cursor Addressable Text
 Reverse Video Off
 Reverse Video On
 Set Cursor Text Attributes
 Set Cursor Text Color Index

Description

The Enter Cursor Addressing Mode function exits Graphics Mode if different from Cursor Addressing Mode.

This function must precede all other cursor addressing functions, such as Cursor Up. This function returns the cursor to the *home* position.

Cursor addressing is meaningful only on CRT devices. It is defined on a character cell grid of rows and columns (rows equals the number of lines on a screen, and columns equals the number of character cells per line). The upper left-hand corner of the screen is row 1, column 1.

See the Open Workstation parameter *work_out*[45] to determine if the device represented by *dev_handle* is a CRT.

Note

This function is applicable only to CRT devices.

3.20 Erase To End Of Line**Class**

Control Functions

Inputs

dev..handle	CGIhandle for a currently open workstation
-------------	--------------------------------------------

Outputs

None

Value Returned

≥ 0	no error
-1	error occurred; actual error can be retrieved by invoking Inquire CGIError

Related Functions

Enter Cursor Addressing Mode
 Erase To End Of Screen
 Inquire Current Cursor Text Address
 Reverse Video Off
 Reverse Video On
 Set Cursor Text Attributes
 Set Cursor Text Color Index

Description

When you are in Cursor Addressing Mode, the Erase To End Of Line function erases from the current cursor position to the end of the line. The line is erased by writing spaces with the current Cursor Text Attributes.

Note

This function is applicable only to CRT devices.

3.21 EraseTo EndOfScreen

Class

ControlFunctions

Inputs

dev_handle CGI handle for a currently open workstation

Outputs

None

Value Returned

≥0 no error

-1 error occurred; actual error can be retrieved by invoking Inquire CGIError

Related Functions

- Enter Cursor Addressing Mode
- Erase To End Of Line
- Inquire Current Cursor Text Address
- Reverse Video Off
- Reverse Video On
- Set Cursor Text Attributes
- Set Cursor Text Color Index

Description

When you are in Cursor Addressing Mode, the Erase To End Of Screen function erases from the current cursor to the end of the screen.

Note

This function is applicable only to CRT devices.

3.22 Escape**Class****Control Functions****Inputs**

<code>dev_handle</code>	CGIhandle for a currently open workstation
<code>esc_code</code>	escape opcode assigned by SCO to the vendor
<code>in_count</code>	number of integers in <i>intin</i> array
<code>intin[]</code>	array containing integer input data (format is vendor specific)

Outputs

<code>intout[]</code>	array containing integer output data (format is vendor specific)
------------------------	------------------------------------------------------------------

Value Returned

≥ 0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

None

Description

This routine passes non-SCO-specified data to a device. This allows an application program to access a device's specialized capability where the device's vendor received a specific escape opcode from SCO. The format of the input and output data should be found in the device vendor's manual.

3.23 Exit Cursor Addressing Mode

Class

Control Functions

Inputs

dev_handle CGIhandle for a currently open workstation

Outputs

None

ValueReturned

≥0 no error

-1 error occurred; actual error can be retrieved by invoking `InquireCGIError`

Related Functions

Enter Cursor Addressing Mode

Description

If already in Cursor Addressing Mode, this function exits Cursor Addressing Mode; otherwise, it does nothing. In order to display graphics, this function must be used to enter Graphics Mode from Cursor Addressing Mode.

3.24 Hard Copy

Class

Control Functions

Inputs

dev_handle CGIhandle for a currently open workstation

Outputs

None

Value Returned

≥0 no error

-1 error occurred; actual error can be retrieved by
invoking Inquire CGIError

Related Functions

None

Description

The hard copy function generates a hard copy. This function is very device-specific and may involve copying the screen to a printer or to another attached hard-copy device.

Note

For example, in the IBM PC environment, GRAPHICS.COM must be executed prior to the CGI application in order for this function to work correctly.

3.25 Inquire Addressable Character Cells

Class

Inquiry Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
-------------------------	---------------------------------------------

Outputs

<code>rows</code>	number of addressable rows on the screen
-------------------	------------------------------------------

<code>columns</code>	number of addressable columns on the screen
----------------------	---------------------------------------------

Value Returned

0	no error
---	----------

-1	error occurred; actual error can be retrieved by invoking <code>InquireCGIError</code>
----	----------------------------------------------------------------------------------------

Related Functions

- Direct Cursor Address
- Enter Cursor Addressing Mode
- Inquire Current Cursor Text Address
- Output Cursor Addressable Text

Description

The Inquire Addressable Character Cells function returns the number of cursor addressable columns and the number of cursor addressable rows on the screen. This function is useful for determining addressable page size. It is applicable only to CRT devices.

3.26 Inquire Alpha Text Capabilities

Class

Inquiry Functions

Inputs

`dev_handle` CGI handle for a currently open workstation

Outputs

`alph_cap[0]` superscript capability flag:

0 no

1 yes

`alph_cap[1]` subscript capability flag:

0 no

1 yes

`alph_cap[2]` underline capability flag:

0 no

1 yes

`alph_cap[3]` overstrike capability flag:

0 no

1 yes

`alph_cap[4]` number of discrete alpha text sizes where size 2 is larger (occupies more area) than size 1, and so on; at least one size must be present

`alph_cap[5]` discrete size index of the default font; this size is dependent on the number of font sizes supported

SCO CGI Programmer's Guide

<code>alph_cap[6]</code>	character positioning capability flag: 0 characters positionable on cell boundaries only 1 characters positionable on a finer grid than a character cell, but not necessarily the same grid as graphics
<code>alph_cap[7]</code>	number of horizontal character cell positions across the display surface in the default font; for a typical CRT or printer that can only place text on cell boundaries, this is 80
<code>alph_cap[8]</code>	number of vertical character cell positions down the display surface in the default font; this is 24 for a typical CRT and 66 for a typical printer that can only place text on cell boundaries
<code>alph_cap[9]</code>	number of horizontal character cell positions represented by the distance specified in <i>alph_cap[13]</i>
<code>alph_cap[10]</code>	number of vertical character cell positions represented by the distance specified in <i>alph_cap[14]</i>
<code>alph_cap[11]</code>	number of horizontal alpha text grids represented by the distance specified in <i>alph_cap[13]</i>
<code>alph_cap[12]</code>	number of vertical alpha text grids represented by the distance specified in <i>alph_cap[14]</i>
<code>alph_cap[13]</code>	width in VDC units of the number of character cells (in the default font) specified in <i>alph_cap[9]</i>
<code>alph_cap[14]</code>	height in VDC units of the number of character cells (in the default font) specified in <i>alph_cap[10]</i>

Value Returned

0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Set Alpha Text Color Index
Set Alpha Text Font And Size
Set Alpha Text Line Spacing
Set Alpha Text Overstrike Mode
Set Alpha Text Position
Set Alpha Text Subscript/Superscript Mode
Set Alpha Text Underline Mode.

Description

The Inquire Alpha Text Capabilities function returns information regarding the alpha text features of the device.

The *alph_cap*[13]/*alph_cap*[9] ratio can be used to determine the width of a character cell, including any roundoff error.

The *alph_cap*[14]/*alph_cap*[10] ratio can be used to determine the height of a character cell, including any roundoff error.

The *alph_cap*[13]/*alph_cap*[11] ratio can be used to determine the width of the alpha text grid, including roundoff error.

The *alph_cap*[14]/*alph_cap*[12] ratio can be used to determine the height of the alpha text grid, including roundoff error.

3.27 Inquire Alpha Text Cell Location

Class

Inquiry Functions

Inputs

dev_handle	CGI handle for a currently open workstation
row	row number of character cell (1 to number of rows)
column	column number of character cell (1 to number of columns)

Outputs

propflag	proportional spacing flag: 0 no 1 yes
x_out	x coordinate of lower left-hand corner of character cell in VDC units
y_out	y coordinate of lower left-hand corner of character cell in VDC units

Value Returned

0	no error
-1	error occurred; actual error can be retrieved by invoking Inquire CGIError

Related Functions

Set Alpha Text Position

Description

The Inquire Alpha Text Cell Location function returns the virtual device coordinates (VDCs) of the lower left-hand corner of the character cell position specified, based on the current font. This allows text to be positioned in a specific column on the output device. Column 1 implies a 0 x position on the display surface, and row 1 implies the maximum y position on the display surface.

If the *propflag* value is 1, the size represented by *x_out* and *y_out* may not represent the selected font. This is the case if the requested font is proportionally spaced because the character size is not constant.

3.28 Inquire Alpha Text Font Capability

Class

Inquiry Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>font_in</code>	requested font (1 to 3 are constant spaced fonts): 1 normal/standard font (the default) 2 bold (always provided for printers) 3 italics 4 proportionally-spaced normal font 5 proportionally-spaced bold 6 proportionally-spaced italics >6 device-dependent
<code>size_in</code>	requested text size (1 to device maximum), where size $n+1$ is larger (occupies more area) than size n

Outputs

<code>font_status[0]</code>	number of horizontal character cell positions across the display surface in this font; if the requested font is not available, <code>font_status[0]</code> is 0
<code>font_status[1]</code>	number of vertical character cell positions down the display surface in this font; if the requested font is not available, <code>font_status[1]</code> is 0 is not available
<code>font_status[2]</code>	number of horizontal character cell positions represented by the distance specified in <code>font_status[5]</code> ; if the requested font is not available, <code>font_status[2]</code> is 0

<code>font_status[3]</code>	number of vertical character cell positions represented by the distance specified in <code>font_status[6]</code> ; if the requested font is not available, <code>font_status[3]</code> is 0
<code>font_status[4]</code>	proportional spacing flag: 0 no 1 yes
<code>font_status[5]</code>	width in VDC units of the number of character cells (in the selected font) specified in <code>font_status[2]</code> ; if the requested font is not available, <code>font_status[5]</code> is 0
<code>font_status[6]</code>	height in VDC units of the number of character cells (in the selected font) specified in <code>font_status[3]</code> ; if the requested font is not available, <code>font_status[6]</code> is 0

Value Returned

>0	font and size available
0	font and size not available
-1	error occurred; actual error can be retrieved by invoking Inquire CGI Error

Related Functions

Set Alpha Text Font And Size

Description

The Inquire Alpha Text Font Capability function inquires the attributes of a particular alpha text font and size, such as availability on this device and height and width.

This function can be used to determine which font best fits specific size requirements without having to alter the currently set text font.

`font_status[0]` is -1 if the selected font is a proportional font, because the character cell size is not constant.

SCO CGI Programmer's Guide

The $font_status[5]/font_status[2]$ ratio can be used to determine the width of a character cell, including any roundoff error. This value is not accurate if the proportional spacing flag is set to 1, because the character cell size is not constant.

The $font_status[6]/font_status[3]$ ratio can be used to determine the height of a character cell, including any roundoff error.

If $font_status[4]$ is 1, the size represented by $font_status[5]$ and $font_status[6]$ may not represent the selected font. This is the case if the requested font is proportionally-spaced.

3.29 Inquire Alpha Text Position**Class**

Inquiry Functions

Inputs

dev_handle	CGI handle for a currently open workstation
------------	---------------------------------------------

Outputs

x_out	x coordinate of text position in VDC units
-------	--------------------------------------------

y_out	y coordinate of text position in VDC units
-------	--------------------------------------------

Value Returned

0	no error
---	----------

-1	error occurred; actual error can be retrieved by invoking Inquire CGIError
----	----------------------------------------------------------------------------

Related Functions

Set Alpha Text Position

Description

The Inquire Alpha Text Position function reports the current alpha text position (returned in 0 to 32767 VDC units). It is assumed that 0,0 is at the lower left-hand corner of the display surface.

3.30 Inquire Alpha Text String Length

Class

Inquiry Functions

Inputs

dev...handle	CGIhandle for a currently open workstation
string	textstring

Outputs

None

Value Returned

≥ 0	string length in VDC units
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Set Alpha Text Font And Size

Description

The Inquire Alpha Text String Length function returns the length of the text string specified in VDC units, based on the current font in use. If a control character (ASCII 0 to 31) appears in the string, it terminates the string, and the string length up to that point is returned.

This function is useful when using proportional fonts because each character is not the same width. It is also useful for doing microjustification between words because multiplication of the width of a character cell in VDC space may produce inaccurate results due to the inherent roundoff error in the character cell size reported back to the programmer.

3.31 Inquire Background Mode**Class**

Inquiry Functions

Inputs

dev_handle

CGI handle for a currently open workstation

Outputs

None

Value Returned ≥ 0

the current background mode:

0 opaque

1 transparent

-1

error occurred; actual error can be retrieved by invoking `InquireCGIError`**Related Functions**

Set Background Mode

Set Writing Mode

Description

The `Inquire Background Mode` function returns the current background mode.

3.32 Inquire Bitmap Formats

Class

Inquiry Functions

Inputs

dev_handle CGIhandle for a currently open workstation

Outputs

None

Value Returned

<0	error occurred; actual error can be retrieved by invoking Inquire CGIError
0	bitmaps not supported on this device
1	only full-depth color bitmaps supported on this device
2	both full-depth and monochrome (mapped) bitmaps supported on this device

Related Functions

Create Bitmap
Set Background Color Index

Description

The Inquire Bitmap Formats function allows the application to determine the level of bitmap support on this particular device.

3.33 Inquire Byte Pixel Array**Class**

Inquiry Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>origin[0]</code>	<i>x</i> coordinate, in VDC units, of one corner of source rectangle
<code>origin[1]</code>	<i>y</i> coordinate, in VDC units, of one corner of source rectangle
<code>width</code>	width (in pixels) of the source pixel array
<code>height</code>	height (in pixels) of the source pixel array

Outputs

<code>valid_width[0]</code>	first valid column index
<code>valid_width[1]</code>	last valid column index
<code>valid_height[0]</code>	first valid row index
<code>valid_height[1]</code>	last valid row index
<code>pixels[]</code>	array (<code>height*width</code>) holding byte pixels from workstation

Value Returned

≥ 0	no error
-1 error	retrieved by invoking <code>Inquire CGIError</code>

Related Functions

- Inquire Integer Pixel Array
- Output Byte Pixel Array
- Output Integer Pixel Array
- Set Color Representation
- Set Color Table

Description

The Inquire Byte Pixel Array function reads (*width*height*) color indices from the currently selected drawing bitmap of the specified device and stores the pixels in the specified byte array.

The specified *origin* determines the location, in VDC units, where the first pixel value is located.

The only difference between the Inquire Byte Pixel Array function and the Inquire Integer Pixel Array function is that each pixel array element of Inquire Byte Pixel Array can represent color indices between 0 and 255 while elements of Inquire Integer Pixel Array can represent color indices between 0 and 65535.

The pixel array is passed as a vector where each row of the pixel array follows the previous row in the vector. The data are passed in the array from lowest row to highest row and from lowest column to highest column within each row. A pixel array with *n* rows and *m* columns is represented as follows:

Array Element	Description
0	row 1, col 1
1	row 1, col 2
.	.
.	.
.	.
m - 1	row 1, col m
m	row 2, col 1
m + 1	row 2, col 2
.	.
.	.
.	.
2*m - 1	row 2, col m
.	.
.	.
.	.
n*m - 1	row n, col m

width is a signed integer that defines how many pixels are in each row. If *width* is greater than 0, the row extends toward increasing *x* from the specified origin. If *width* is less than 0, the row extends toward decreasing *x* from the specified origin.

height is a signed integer that defines how many rows of pixels there are. If *height* is greater than 0, the rows extend towards increasing *y* from the specified origin. If *height* is less than 0, the rows extend towards decreasing *y* from the specified origin.

Pixels are read a row at a time, beginning with the row specified by the parameter *origin*.

If $width * height$ is zero (0), no pixels are read.

valid_width and *valid_height* specify the first and last values of $abs(width)$ and $abs(height)$ that contain defined color indices (pixels). If the entire array contains defined color indices, then $valid_width = (0, abs(width)-1)$ and $valid_height = (0, abs(height)-1)$. *valid_width* and *valid_height* provide a means to specify the subregion of the pixel array that has been used to store the pixels.

For the simplest applications of this function, the *valid_width* parameter is $(0, abs(width)-1)$ and the *valid_height* parameter is $(0, abs(height)-1)$.

No pixels are returned if either of the following conditions exist in the output parameters *valid_width* and *valid_height*:

$valid_width[1] < valid_width[0]$
(last column less than first column)

or

$valid_height[1] < valid_height[0]$
(last row less than first row)

The clipping rectangle has no effect on pixel arrays. Pixel arrays are limited by the physical limits of the selected bitmap.

The output of the Inquire Byte Pixel Array function is limited to the intersection of the specified input rectangle (*origin*, *width*, and *height*) and the extents of the currently selected drawing bitmap. If the requested *width* or *height* is larger than the current bitmap, or the specified *origin* is outside of the current bitmaps extents, the output parameters *valid_width* and *valid_height* will be set to indicate which rows and columns of the source rectangular area have been returned in the pixel array.

An error is returned if the current workstation's color indices cannot be represented in a byte (color indices greater than 255).

3.34 Inquire Cell Array

Class

Inquiry Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>xy[0]</code>	<i>x</i> coordinate of lower left-hand corner in VDC units
<code>xy[1]</code>	<i>y</i> coordinate of lower left-hand corner in VDC units
<code>xy[2]</code>	<i>x</i> coordinate of upper right-hand corner in VDC units
<code>xy[3]</code>	<i>y</i> coordinate of upper right-hand corner in VDC units
<code>row_length</code>	length of each row in color index array
<code>num_rows</code>	number of rows in color index array

Outputs

<code>el_per_row</code>	number of elements used in each row of color index array
<code>rows_used</code>	number of rows used in color index array
<code>status</code>	invalid value flag: 0 if no errors 1 if a color value could not be determined for some pixel
<code>colors</code>	color index array (stored one row at a time) -1 indicates that a color index could not be determined for that particular pixel

Value Returned

0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Inquire Color representation
Open Workstation
Output Cell Array
Set Color Representation
Set Color Table

Description

The `Inquire Cell Array` function returns color indices one row at a time, starting from the top of the rectangular area and proceeding downward. See the `Output Cell Array` function for more information about how the rectangular area is divided.

See the `Open Workstation` parameter `work_out[38]` to determine if device represented by `dev_handle` is capable of cell array operations.

3.35 Inquire CGIError

Class

Inquiry Functions

Inputs

None

Outputs

None

Value Returned

0	no error
<0	code number of the error from the immediately preceding CGIfunction call

Related Functions

None

Description

The Inquire CGIError function returns the number of the actual error last encountered. It should be called when the function value returned by any other function is -1. See Appendix A, **Error Codes**, for the complete list of error codes.

3.36 Inquire Clip Rectangle**Class**

Inquiry Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
-------------------------	---------------------------------------------

Outputs

<code>clip_rectangle[0]</code>	<i>x</i> coordinate, in VDC units, of the first end point of a diagonal that describes the current clipping rectangle
--------------------------------	-----------------------------------------------------------------------------------------------------------------------

<code>clip_rectangle[1]</code>	<i>y</i> coordinate, in VDC units, of the first end point of a diagonal that describes the current clipping rectangle
--------------------------------	-----------------------------------------------------------------------------------------------------------------------

<code>clip_rectangle[2]</code>	<i>x</i> coordinate, in VDC units, of the second end point of a diagonal that describes the current clipping rectangle
--------------------------------	------------------------------------------------------------------------------------------------------------------------

<code>clip_rectangle[3]</code>	<i>y</i> coordinate, in VDC units, of the second end point of a diagonal that describes the current clipping rectangle
--------------------------------	------------------------------------------------------------------------------------------------------------------------

Value Returned

≥ 0	no error
----------	----------

-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>
----	-----------------------------------------------------------------------------------------

Related Functions

Set Clip Rectangle

Description

The Inquire Clip Rectangle function reports the currently defined clip rectangle in VDC coordinates. The returned canonical rectangle is defined by the lower left and upper right corners. Given that there are typically many virtual device coordinates (VDCs) that map (translate) to a single device coordinate, a canonical rectangle is defined as follows:

- the lower left corner is the minimum VDC that translates to a specific device coordinate that is the lower left corner of the rectangle on the display surface;
- the upper right corner is the maximum VDC that translates to a specific device coordinate that is the upper right corner of the rectangle on the display surface.

The default clip rectangle at open workstation is -32768 to 32767 in both x and y regardless of the open workstation transform mode.

3.37 Inquire Color Representation

Class

Inquiry Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>ind_in</code>	requested color index
<code>set_flag</code>	set or realized flag:
	0 set; return requested color values
	1 realized; return color values realized on device

Outputs

<code>rgb[0]</code>	red intensity; in tenths of percent 0-1000
<code>rgb[1]</code>	green intensity; in tenths of percent 0-1000
<code>rgb[2]</code>	blue intensity; in tenths of percent 0-1000

Value Returned

≥ 0	selected color index
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Open Workstation
Set Color Representation
Set Color Table

Description

The `Inquire Color Representation` function allows inquiry of the color associated with a given index.

SCO CGI Programmer's Guide

If an index outside of the device's capability is requested, the index closest to it is used for the inquiry.

See the Open Workstation parameters *work_out*[13] and *work_out*[39] for device-dependent information regarding this function.

3.38 Inquire Current Cursor Text Address**Class**

Inquiry Functions

Inputs

dev_handle	CGI handle for a currently open workstation
------------	---------------------------------------------

Outputs

row	row number (1 to number of rows)
column	column number (1 to number of columns)

Value Returned

0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Direct Cursor Address
 Enter Cursor Addressing Mode
 Output Cursor Addressable Text

Description

When you are in Cursor Addressing Mode, this function returns the current cursor position.

Note

This function is applicable only to CRT devices.

3.39 Inquire Current Fill Area Attributes

Class

Inquiry Functions

Inputs

dev_handle CGI handle for a currently open workstation

Outputs

attrib[0] current fill area interior style:

- 0 hollow
- 1 solid
- 2 pattern
- 3 hatch
- 4 bitmap

attrib[1] current fill area color index

attrib[2] current fill area style index:

- 1 narrow spaced +45° lines
- 2 medium spaced +45° lines
- 3 widely spaced +45° lines
- 4 narrow spaced +45° and -45° lines
- 5 medium spaced +45° and -45° lines
- 6 widely spaced +45° and -45° lines
- >6 device-dependent

attrib[3] current writing mode (see the Set Writing Mode function for description)

ValueReturned

0	no error
-1	error occurred; actual error can be retrieved by invoking Inquire CGIError

Related Functions

Output Bar
Output Circle
Output Ellipse
Output Elliptical Pie Slice
Output Filled Area
Output Pie Slice
Set Fill Color Index
Set Fill Interior Style
Set Fill Style Index
Set WritingMode

Description

The Inquire Current Fill Area Attributes function reports the current setting of all attributes that affect filled areas, bars, pie slices, ellipses, elliptical arcs, elliptical pie slices, and circles. These include interior style, fill color and fill style index.

3.40 Inquire Current Graphics Text Attributes

Class

Inquiry Functions

Inputs

dev_handle CGI handle for a currently open workstation

Outputs

attrib[0]	current graphics text font
attrib[1]	current graphics text color
attrib[2]	current angle of rotation of text baseline (in tenths of degrees 0 to 3599)
attrib[3]	current horizontal alignment: 0 left justified (default) 1 centerjustified 2 right justified
attrib[4]	current vertical alignment: 0 base 1 half 2 cap 3 bottom 4 top
attrib[5]	current writing mode (see the Set Writing Mode function for description)

attrib[6]	current character width in VDC units
attrib[7]	current character height in VDC units
attrib[8]	current character cell width in VDC units
attrib[9]	current character cell height in VDC units

Value Returned

≥ 0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Output Graphics Text
Set Graphics Text Alignment
Set Graphics Text Character Height
Set Graphics Text Color Index
Set Graphics Text Font
Set Graphics Text String Baseline Rotation
Set Writing Mode

Description

The `Inquire Current Graphics Text Attributes` function reports the current setting of all attributes that affect graphics text.

Refer to the `Set Graphics Text Character Height` function for a description of the font coordinate system.

3.41 Inquire Current Line Attributes

Class

Inquiry Functions

Inputs

dev_handle CGI handle for a currently open workstation

Outputs

attrib[0] current line type:
 0 user-defined
 1 solid
 2 long dashed
 3 dotted
 4 dashed-dotted
 5 medium dashed
 6 dashed with two dots
 7 short dashed
 >7 device-dependent

attrib[1] current line color index

attrib[2] current writing mode (see the Set Writing Mode
 function for description)

attrib[3] current line width in VDC units

Value Returned

0	no error
-1	error occurred; actual error can be retrieved by invoking Inquire CGIError

Related Functions

Output Arc
Output Elliptical Arc
Output Polyline
Set Line Color Index
Set Line Type
Set Line Width
Set Writing Mode
Set User Line Type

Description

The Inquire Current Line Attributes function reports the current setting of all attributes that affect polylines and arcs, such as line type, line color, line width and writing mode.

3.42 Inquire Current Marker Attributes

Class

Inquiry Functions

Inputs

`dev_handle` CGI handle for a currently open workstation

Outputs

`attrib[0]` currentmarker type:

- 0 user-defined
- 1 .
- 2 +
- 3 *
- 4
- 5 X
- 6

>6 device-dependent

`attrib[1]` current marker color index

`attrib[2]` current writing mode (see the Set Writing Mode function for description)

`attrib[3]` current marker height in VDC units

Value Returned

0 no error

-1 error occurred; actual error can be retrieved by invoking `Inquire CGIError`

Related Functions

Output Polymarker
Set Marker Color Index
Set Marker Height
Set Marker Type
Set Writing Mode

Description

The Inquire Current Marker Attributes function reports the current setting of all attributes that affect polymarkers, such as marker type, marker color, and marker height.

3.43 Inquire Curs or Description

Class

Inquiry Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>cursor_handle</code>	handle of cursor description to be inquired about

Outputs

<code>data_bitmap</code>	bitmap handle of the bitmap used as the XOR (data) bitmap for the cursor definition
<code>mask_bitmap</code>	bitmap handle of the bitmap used as the NAND (mask) bitmap for the cursor definition
<code>x</code>	<i>x</i> coordinate of the reference point (hot spot) in the data and mask bitmaps; this point will be aligned at the selected input coordinate
<code>y</code>	<i>y</i> coordinate of the reference point (hot spot) in the data and mask bitmaps; this point will be aligned at the selected input coordinate

Value Returned

≥ 0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Create Cursor
Inquire Graphics Input Cursor
Select Graphics Input Cursor

Description

The `Inquire Cursor Description` function returns the information that was used to define the specified *cursor_handle*.

The predefined cursors are not inquirable, and they may not be redefined.



3.44 Inquire deferralmode

Class

Inquiry Functions

Inputs

dev_handle CGI handle for a currently open workstation

Outputs

None

Value Returned

≥0 deferralmode:
0 at some time in the future (ASTI) (default)
1 before thenextuserinteraction (BNI)
2 assoon as possible (ASAP)

-1 error occurred; actual error can be retrieved by
 invoking Inquire CGIError

Related Functions

Set Deferral Mode
Reset To Defaults

Description

This function reports the currently selected deferral mode.

3.45 Inquire Displayable Bitmaps

Class

Inquiry Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>n</code>	number of array elements available in <i>displays</i>

Outputs

<code>displays[0]</code>	number of displayable graphics bitmaps supported by this device
<code>displays[1]</code>	number of displayable cursor text maps supported by this device
<code>displays[2]</code>	requested viewing mode: 0 a bitmap and a cursor text map cannot be viewed at the same time on this device 1 bitmap and cursor text map viewing may be mixed arbitrarily on this device
<code>displays[3]</code>	number of simultaneously displayable graphics bitmaps supported by this device
<code>displays[4]</code>	number of simultaneously displayable cursor text maps supported by this device

Value Returned

≥ 0	number of array elements needed to return the entire <i>displays</i> list
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

SelectDrawingBitmap

Description

Some display devices provide the ability to display different bitmaps or different cursor text maps. These different display maps are sometimes called pages.

The `Inquire Displayable Bitmaps` function returns the number of displayable maps available for each mode and the combinations allowed for viewing of graphics bitmaps and cursor text maps.

Note that the display index is zero-based, in the range $0..(displays[0]-1)$.

If no error is encountered, this function always returns the number of array elements integers necessary to obtain all of the associated attributes. The intent is to provide the application with the information necessary to dynamically allocate the memory necessary for the storage of the associated attribute array. A typical sequence might be:

1. Inquire with $n=0$;
2. dynamically allocate an array; the length is the value returned by the previous function call;
3. inquire using the dynamically allocated array as the output parameter.

3.46 Inquire Drawing Bitmap**Class****Inquiry Functions****Inputs**

<code>dev_handle</code>	CGI handle for a currently open workstation
-------------------------	---------------------------------------------

Outputs

<code>bitmap_handle</code>	handle of currently selected bitmap
<code>xy[0]</code>	x coordinate of lower left corner of currently selected bitmap in VDC units
<code>xy[1]</code>	y coordinate of lower left corner of currently selected bitmap in VDC units
<code>xy[2]</code>	x coordinate of upper right corner of currently selected bitmap in VDC units
<code>xy[3]</code>	y coordinate of upper right corner of currently selected bitmap in VDC units

Value Returned

≥ 0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Create Bitmap
Select Drawing Bitmap

Description

The Inquire Drawing Bitmap function returns in *bitmap_handle* the currently selected bitmap for this workstation and in *xy* the bitmap's extent rectangle in VDC coordinates. The returned canonical rectangle is defined by the lower left and upper right corners. Given that there are typically many virtual device coordinates (VDCs) that map (translate) to a single device coordinate a canonical rectangle is defined as follows:

- the lower left corner is the minimum VDC that translates to a specific device coordinate that is the lower left corner of the rectangle on the display surface;
- the upper right corner is the maximum VDC that translates to a specific device coordinate that is the upper right corner of the rectangle on the display surface.

3.47 Inquire FillArea Representation

Class

Inquiry Functions

Inputs

`dev_handle` CGIhandle for a currently open workstation
`n` the number of elements available to *attrib*

Outputs

`attrib[0]` current fill area interior style:
 0 hollow
 1 solid
 2 pattern
 3 hatch
 4 bitmap

`attrib[1]` current fill area color index; meaningless if current fill area interior style is bitmap (*attrib[0]=4*)

`attrib[2]` current fill area style; meaning depends on fill area interior style (*attrib[0]*):

<code>attrib[0]</code> value	<code>attrib[2]</code> value
hollow	meaningless
solid	meaningless
pattern	pattern index
hatch	hatch index
bitmap	bitmap handle

Value Returned

≥ 0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Output Bar
Output Circle
Output Ellipse
Output Elliptical Pie Slice
Output Filled Area
Output Pie Slice
Set Fill Area Representation
Set Fill Color Index
Set Fill Interior Style
Set Fill Style Index

Description

The `Inquire Fill Area Representation` function returns the current fill area attributes.

If the parameter n is less than the total number of attributes, only the first n attributes will be returned.

If no error is encountered, this function always returns the number of elements necessary to obtain all of the associated attributes. The intent is to provide the application with the information necessary to dynamically allocate the memory necessary for the storage of the associated attribute array. A typical sequence might be:

1. `Inquire` with $n=0$;
2. dynamically allocate an array; the length is the value returned by the previous function call;
3. `inquire` using the dynamically allocated array as the output parameter.

3.48 Inquire Graphics Input Cursor

Class

Inquiry Functions

Inputs

`dev_handle` CGI handle for a currently open workstation

Outputs

`cursor_handle` *cursor_handle* (or index) of the currently selected graphics input cursor

Value Returned

≥ 0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Create Cursor
 Display Graphics Input Cursor
 Inquire Cursor Description
 Remove Graphics Input Cursor
 Select Graphics Input Cursor

Description

The Inquire Graphics Input Cursor function returns the *cursor_handle* of the currently selected graphics cursor. There are six predefined cursors with indices 0 through 5; *cursor_handles* other than the predefined indices are expected to be cursor handles returned from Create Cursor. The *cursor_handle* may be used with the Inquire Cursor Description function to inquire the cursor's description if the specified cursor is user-defined. See Figure 3-9, *Graphics Input Cursors*, in the Select Graphics Input Cursor function.

3.49 Inquire Graphics Text Extent

Class

Inquiry Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>xin</code>	<i>x</i> coordinate for the alignment point in VDC units
<code>yin</code>	<i>y</i> coordinate for the alignment point in VDC units
<code>string</code>	string to determine extent rectangle for

Outputs

<code>xcon</code>	<i>x</i> coordinate for the concatenation point in VDC units
<code>ycon</code>	<i>y</i> coordinate for the concatenation point in VDC units
<code>rectangle[0]</code>	<i>x</i> component of the first corner of text rectangle in VDC units
<code>rectangle[1]</code>	<i>y</i> component of the first corner of text rectangle in VDC units
<code>rectangle[2]</code>	<i>x</i> component of the second corner of text rectangle in VDC units
<code>rectangle[3]</code>	<i>y</i> component of the second corner of text rectangle in VDC units
<code>rectangle[4]</code>	<i>x</i> component of the third corner of text rectangle in VDC units
<code>rectangle[5]</code>	<i>y</i> component of the third corner of text rectangle in VDC units
<code>rectangle[6]</code>	<i>x</i> component of the fourth corner of text rectangle in VDC units

rectangle[7]	y component of the fourth corner of text rectangle in VDC units
bool[0]	first corner valid flag
bool[1]	second corner valid flag
bool[2]	third corner valid flag
bool[3]	fourth corner valid flag
bool[4]	concatenation point valid flag

Value Returned

≥ 0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Inquire Graphics Text Representation
 Output Graphics Text
 Set Graphics Text Alignment
 Set Graphics Text Character Height
 Set Graphics Text Font
 Set Graphics Text Representation
 Set Graphics Text String Baseline Rotation

Description

The extent of the specified character string is computed using the current settings for Text Font, Character Height, Text Baseline Rotation, and Text Alignment.

The points returned in the text extent rectangle are defined to be the four corners of the rectangle taken in a clockwise direction. The first point is the lower left corner of the rectangle with no text rotation applied.

The concatenation point can be used as the origin of a subsequent text output element for the concatenation of character strings with the same font, height, baseline rotation and alignment. The concatenation point that is returned for horizontally center aligned text is the alignment point, therefore in this case text concatenation is not possible by using the concatenation point.

SCO CGI Programmer's Guide

The *bool* output parameters are flags that indicate the validity of the returned rectangle and concatenation point: 0 = outside the visible region; 1 = inside the visible region.

3.50 Inquire Graphics Text Font Character

Class

Inquiry Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>character</code>	the character for which information is requested
<code>n</code>	total number of elements that will fit in the array <i>description</i>

Outputs

<code>description[0]</code>	character body width in VDC units
<code>description[1]</code>	character body height in VDC units
<code>description[2]</code>	cell width in VDC units
<code>description[3]</code>	cell height in VDC units
<code>description[4]</code>	horizontal offset of start of character body in relation to cell in VDC units

Value Returned

≥ 0	the number of array elements needed to obtain the entire <i>description</i> list
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Inquire Graphics Text Representation
 Output Graphics Text
 Set Graphics Text Character Height
 Set Graphics Text Font
 Set Graphics Text Representation
 Set Graphics Text String Baseline Rotation

Description

The **Inquire Graphics Text Font Character** function permits the application to determine the measurements of an individual character in the current text font (as set by the **Set Graphics Text Font** function). The VDC units returned apply to the currently specified text attributes, such as Text Height. See Figure 3-14, *The Font Coordinate System*, in the **Set Graphics Text Character Height** function for an illustration.

If the input parameter n is less than the function's returned value, the output data is truncated to the specified length n .

If no error is encountered, this function always returns the number of array elements necessary to obtain all of the associated attributes. The intent is to provide the application with the information necessary to dynamically allocate the memory necessary for the storage of the associated attribute array. A typical sequence might be:

1. Inquire with $n=0$;
2. dynamically allocate an array; the length is the value returned by the previous function call;
3. inquire using the dynamically allocated array as the output parameter.

3.51 Inquire Graphics TextFont Description

Class

Inquiry Functions

Inputs

`dev_handle` CGIhandle for a currently open workstation
`n` number of elements in the array *description*

Outputs

`description[0]` font file version
`description[1]` font type:
 0 software bitmap
 1 hardware bitmap
 2 software outline
 3 hardware outline

`description[2]` the *nominal* character number; the number of
 the prototype character around which the font is
 designed

`description[3]` spacing type:
 0 monospaced
 1 proportionally spaced

`description[4]` character set:
 0 IBMPC extended ASCII
 1 X3.4 ASCII
 2 special

SCO CGI Programmer's Guide

description[5]	minimum character number; this is the lowest defined character in the font
description[6]	maximum character number
description[7]	undefined character number; the character that is substituted for requests for non-defined characters
description[8]	number of character sizes available
description[9]	scaling technique used: 0 multiple sized bitmaps 1 pixel replication 2 uniformly scalable
description[10]	nominal horizontal resolution (dots per inch) that was used to digitize this font
description[11]	nominal vertical resolution (dots per inch) that was used to digitize this font
description[12]	kerning flag: 0 no kerning 1 uses kerning
description[13]	true if the font is suitable for 0 or 180° rotation, else false
description[14]	true if the font is suitable for 90 or 270° rotation, else false
description[15]	smallest character height available, in VDC units
description[16]	largest character height available, in VDC units
font_name	up to 19 ASCII character font name plus a null terminator; total possible 20 bytes

Value Returned

- | | |
|----------|-----------------------------------------------------------------------------------------|
| ≥ 0 | the number of array elements needed to obtain the entire <i>description</i> list |
| -1 | error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code> |

Related Functions

Inquire Graphics Text Representation
 Output Graphics Text
 Set Graphics Text Character Height
 Set Graphics Text Font
 Set Graphics Text Representation
 Set Graphics Text String Baseline Rotation

Description

The `Inquire Graphics Text Font Description` function reports the font description for the current font index (as set with the `Set Graphics Text Font` function).

If the input parameter n is less than the function's returned value, then the output data is truncated to the specified length n .

If no error is encountered, this function always returns the number of array elements necessary to obtain all of the associated attributes. The intent is to provide the application with the information necessary to dynamically allocate the memory necessary for the storage of the associated attribute array. A typical sequence might be:

1. `Inquire` with $n=0$;
2. dynamically allocate an array; the length is the value returned by the previous function call;
3. `inquire` using the dynamically allocated array as the output parameter.

3.52 Inquire Graphics TextFontMetrics

Class

Inquiry Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>n</code>	number of elements in the output array <i>metrics</i>

Outputs

<code>metrics[0]</code>	nominal character body width in VDC units
<code>metrics[1]</code>	nominal character body height in VDC units; the distance from the baseline to the capline
<code>metrics[2]</code>	nominal cell width in VDC units
<code>metrics[3]</code>	nominal cell height in VDC units; the distance from the bottom line to the top line
<code>metrics[4]</code>	maximum cell width in VDC units
<code>metrics[5]</code>	maximum cell height in VDC units
<code>metrics[6]</code>	distance from cell left edge to character rectangle in VDC units
<code>metrics[7]</code>	distance from cell bottom to baseline in VDC units
<code>metrics[8]</code>	space width in VDC units
<code>metrics[9]</code>	distance from baseline to halfline in VDC units

Value Returned

≥ 0	the number of array elements needed to obtain the entire <i>description</i> list
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Inquire Graphics Text Representation
 Output Graphics Text
 Set Graphics Text Font
 Set Graphics Text Representation

Description

The Inquire Graphics Text Font Metrics function permits the application to determine the measurements of the current text font (as set with the Set Graphics Text Font function). The VDC units returned are modified by the currently specified text attributes, such as Text Height and Text Rotation. See Figure 3-14, *The Font Coordinate System*, in the Set Graphics Text Character Height function for an illustration.

If the input parameter n is less than the function's returned value, the output data is truncated to the specified length, n .

If no error is encountered, this function always returns the number of array elements necessary to obtain all of the associated attributes. The intent is to provide the application with the information necessary to dynamically allocate the memory necessary for the storage of the associated attribute array. A typical sequence might be:

1. Inquire with $n=0$;
2. dynamically allocate an array; the length is the value returned by the previous function call;
3. inquire using the dynamically allocated array as the output parameter.

3.53 Inquire Graphics Text Representation

Class

Inquiry Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>n</code>	number of elements to fill in the array <i>attrib</i>

Outputs

<code>attrib[0]</code>	current graphics text font
<code>attrib[1]</code>	current graphics text color
<code>attrib[2]</code>	current graphics text baseline angle (in tenths of degrees 0 to 3599)
<code>attrib[3]</code>	current horizontal alignment: 0 left 1 center 2 right
<code>attrib[4]</code>	current vertical alignment: 0 base 1 half 2 cap 3 bottom 4 top
<code>attrib[5]</code>	current nominal character width in VDC units

attrib[6]	current nominal character height in VDC units
attrib[7]	current nominal character cell width in VDC units
attrib[8]	current nominal character cell height in VDC units

Value Returned

≥ 0	number of elements in the array <i>attrib</i>
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Output Graphics Text
 Set Graphics Text Alignment
 Set Graphics Text Character Height
 Set Graphics Text Color Index
 Set Graphics Text Font
 Set Graphics Text Representation
 Set Graphics Text String Baseline Rotation

Description

The `Inquire Graphics Text Representation` function reports the current graphics text representation (current text attributes).

If the parameter *n* is less than the total number of attributes, only the first *n* attributes will be returned.

If no error is encountered, this function always returns the number of array elements integers necessary to obtain all of the associated attributes. The intent is to provide the application with the information necessary to dynamically allocate the memory necessary for the storage of the associated attribute array. A typical sequence might be:

1. `Inquire` with *n*=0;
2. dynamically allocate an array; the length is the value returned by the previous function call;
3. `inquire` using the dynamically allocated array as the output parameter.

Refer to the Set Graphics Text Character Height function for a description of the font coordinate system.

3.54 Inquire Input Extent

Class

InquiryFunctions

Inputs

dev_handle CGIhandle for a currently open workstation

Outputs

rectangle[0] *x* coordinate of the first corner of the input device's extent in VDC units

rectangle[1] *y* coordinate of the first corner of the input device's extent in VDC units

rectangle[2] *x* coordinate of the second corner of the input device's extent in VDC units

rectangle[3] *y* coordinate of the second corner of the input device's extent in VDC units

Value Returned

≥ 0 no error

-1 error occurred; actual error can be retrieved by invoking Inquire CGIError

Related Functions

Request Locator

Sample Locator

Set Input Extent

Description

The Inquire Input Extent function returns the current extents of the area on the output echo device to which all input coordinates are restricted.

3.55 Inquire Integer Pixel Array

Class

Inquiry Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>origin[0]</code>	x coordinate, in VDC units, of one corner of source rectangle
<code>origin[1]</code>	y coordinate, in VDC units, of one corner of source rectangle
<code>width</code>	width (in pixels) of the source rectangular area
<code>height</code>	height (in pixels) of the source rectangular area

Outputs

<code>valid_width[0]</code>	first valid column index
<code>valid_width[1]</code>	last valid column index
<code>valid_height[0]</code>	first valid row index
<code>valid_height[1]</code>	last valid row index
<code>pixels[]</code>	array (height*width) holding integer pixels from workstation

Value Returned

≥ 0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Inquire Byte Pixel Array
Output Byte Pixel Array
Output Integer Pixel Array
Set Color Representation
Set Color Table

Description

The Inquire Integer Pixel Array function reads (*width*height*) color indices from the specified output device and stores the pixels in the specified array.

The specified *origin* determines the location, in VDC space, at which the first pixel value is located.

The only difference between the Inquire Byte Pixel Array function and the Inquire Integer Pixel Array function is that each pixel array element of Inquire Byte Pixel Array can represent color indices between 0 and 255 and Inquire Integer Pixel Array can represent color indices between 0 and 65535.

The clipping rectangle has no effect on pixel arrays. Pixel arrays are limited by the physical limits of the selected bitmap.

3.56 Inquire Line Representation

Class

Inquiry Functions

Inputs

<code>dev_handle</code>	CGIhandle for a currently open workstation
<code>n</code>	the number of elements in the output array <i>attrib</i>

Outputs

<code>attrib[0]</code>	line type: 0 user-specified; see <i>attrib[2]</i> , <i>attrib[3]</i> below 1 solid 2 long dashed 3 dotted 4 dashed-dotted 5 medium dashed 6 dashed with two dots 7 short dashed >7 device dependent
------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<code>attrib[1]</code>	line color index
------------------------	------------------

<code>attrib[2]</code> , <code>attrib[3]</code>	this parameter specifies a 32-bit user defined line type; <i>attrib[2]</i> is the low-order half and <i>attrib[3]</i> is the high-order half
-------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------

<code>attrib[4]</code>	line width in VDC units
<code>attrib[5]</code>	line cross section; see the Set Line Cross Section function
0	solid
1- <i>n</i>	fill style index, as when fill style interior is pattern
> <i>n</i>	bitmap handle of user-defined cross section

Value Returned

≥ 0	the number of array elements needed to obtain the entire description list
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Output Arc
 Output Elliptical Arc
 Output Polyline
 Set Line Color Index
 Set Line Cross Section
 Set Line Type
 Set Line Width
 Set Line Representation
 Set User Line Type

Description

The `Inquire Line Representation` function returns the current line attributes.

If the parameter *n* is less than the total number of attributes, only the first *n* attributes will be returned. The output parameters `attrib[2]` and `attrib[3]` are valid only if `attrib[0]` is 0. It is the responsibility of the individual language bindings to do whatever word or byte reordering is necessary to accomplish the above assignment.

If no error is encountered, this function always returns the number of array elements necessary to obtain all of the associated attributes. The intent is to provide the application with the information necessary to dynamically allocate the memory necessary for the storage of the associated attribute array. A typical sequence might be:

1. Inquire with $n=0$;
2. dynamically allocate an array; the length is the value returned by the previous function call;
3. inquire using the dynamically allocated array as the output parameter.

3.57 Inquire Marker Representation

Class

Inquiry Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>n</code>	the number of elements in the output array <i>attrib</i>

Outputs

<code>attrib[0]</code>	marker type
	0 user-specified; see <code>attrib[3]</code>
	1 .
	2 +
	3 *
	4 <input type="checkbox"/>
	5 X
	6 \diamond
	>6 device dependent
<code>attrib[1]</code>	marker color index
<code>attrib[2]</code>	marker height in VDC units; used only if <code>attrib[0]</code> is not 0
<code>attrib[3]</code>	handle of bitmap for user-specified marker; valid only if <code>attrib[0]</code> is 0
<code>attrib[4]</code>	x coordinate of the marker hot spot; used only if <code>attrib[0]</code> is 0
<code>attrib[5]</code>	y coordinate of the marker hot spot; used only if <code>attrib[0]</code> is 0

Value Returned

≥ 0	the number of array elements needed to obtain the entire <i>description</i> list
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Inquire Marker Type
Output Polymarker
Set Marker Color Index
Set Marker Height
Set Marker Representation
Set Marker Type

Description

The `Inquire Marker Representation` function returns the current attributes used in polymarker primitives.

If the parameter n is less than the total number of attributes, only the first n attributes will be returned.

The output parameters `attrib[2]` and `attrib[3]` are valid only if `attrib[0]` is 0. It is the responsibility of the individual language bindings to do whatever word or byte reordering is necessary to accomplish the above assignment.

If no error is encountered, this function always returns the number of array elements necessary to obtain all of the associated attributes. The intent is to provide the application with the information necessary to dynamically allocate the memory necessary for the storage of the associated attribute array. A typical sequence might be:

1. `Inquire` with $n=0$;
2. dynamically allocate an array; the length is the value returned by the previous function call;
3. `inquire` using the dynamically allocated array as the output parameter.

3.58 Inquire OptimumPatternSize**Class**

Inquiry Functions

Inputs

dev_handle	CGIhandle for a currently open workstation
------------	--------------------------------------------

Outputs

xy[0]	optimum size for pattern for this device in <i>x</i> in device units
xy[1]	optimum size for pattern for this device in <i>y</i> in device units

Value Returned

≥0	no error
-1	error occurred; actual error can be retrieved by invoking Inquire CGIError

Related Functions

Set Fill Area Representation

Description

The Inquire Optimum Pattern Size function returns the pattern size that will provide optimum performance for this device. Other pattern sizes may be used for pattern extents, but may result in significant performance penalties.

Bitblt hardware is often tailored to a particular pattern size for performance reasons. This inquiry allows the application to tailor pattern size for optimum performance.

SCO CGI Programmer's Guide

A returned size of zero in x or y means that no size has any particular advantage over any other along that axis. Thus a returned size of $(8,0)$ means that stripes eight device units wide and of arbitrary height are the optimum fill pattern.

The pattern can be set with the Set Fill Area Representation function.

3.59 Load CGI**Class**

Control Functions

Inputs

<i>where</i>	requested location in memory for CGI
--------------	--------------------------------------

<i>bytes_avail</i>	number of contiguous bytes available
--------------------	--------------------------------------

Outputs

None

Value Returned

>0	number of bytes required to load the CGI and drivers
----	------------------------------------------------------

-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>
----	-----------------------------------------------------------------------------------------

Related Functions

Remove CGI

Description

The Load CGI function returns the number of bytes required to hold the entire CGI and its drivers. If *bytes_avail* allows room for the CGI and its drivers, the CGI is loaded at *where*. It will not load the CGI if there is not enough room.

It is suggested that an application invoke the function once with *bytes_avail* equal to zero to find out how much room to allocate, then invoke it again with the allocated space.

Note

This function is valid only for DOS. In the interest of code portability, this function is present in the XENIX libraries and will return -1.

The following pseudocode example demonstrates the correct use of this function in the DOS environment.

```
{Call Load Cgi first time with bytes = 0}
bytes_avail = 0

{Find out how many bytes are needed for the load}
bytes_avail = LOAD_CGI(where, bytes_avail)

{Check for error}
if (bytes_avail == -1) then
    {Flag that application did not do the load}
    cgi_loaded_by_me = false
else
    {Allocate memory for the load}
    where = ALLOCATION_ROUTINE (bytes_avail)
    {Check returned pointer}
    if (where is null) then
        {Not enough memory to load CGI. Abort}
    else
        bytes_avail = LOAD_CGI(where, bytes_avail)
        {Check for error}
        if (bytes_avail == -1) then
            {Handle returned error and abort}
        else
            cgi_loaded_by_me = true

    .
    .
    .
    {Just before termination}
    {Fall through and terminate}
if (cgi_loaded_by_me is true) then
    CGI_KILL()
```

The above algorithm will allow an application to run whether or not the controller is loaded transiently (see Appendix A, **GSS*CGI Installation**, of the *GSS*CGI Programmer's Guide for DOS* for more information regarding Transient CGI).

3.60 Message

Class

Control Functions

Inputs

dev_handle	metafile device handle
message	textstring
wait	pause indicator: 0 if no response required 1 if pause after issuing message and wait for a response

Outputs

None

Value Returned

0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Application Data

Description

The `Message` function places a text string in the metafile which will be displayed by the metafile interpreter as an operator message. It appears on the console in a device-dependent position.

The pause indicator controls whether the metafile interpreter will pause for a device-dependent response or continue.

Note

This function is valid only for the Computer Graphics Metafile driver.

3.61 Open Workstation

Class

Control Functions

Inputs

work_in[0]	coordinate transformation mode flag; determines how to transform VDC space to device coordinates (see Description below for more information): 0 full screen 1 preserve aspect ratio 2 device units 3 short axis
work_in[1]	linetype
work_in[2]	line color index
work_in[3]	Marker type
work_in[4]	Marker color index
work_in[5]	graphics text font
work_in[6]	graphics text color index
work_in[7]	fill interior style
work_in[8]	fill style index
work_in[9]	fill color index
work_in[10]	prompting flag for controlling screen on plotters: 0 do not display device-dependent prompts to the logical message device 1 display device-dependent prompts to the logical message device

`work_in[11]` to `work_in[18]`
 workstation identifier, for example, device driver logical name

Outputs

`dev_handle` device handle associated with the workstation identifier (`work_in[11]` to `work_in[18]`)

`work_out[0]` maximum addressable width of screen/plotter in rasters/steps assuming a 0 start point; for example, a resolution of 640 implies an addressable area of 0-639, so `work_out[0]=639`

`work_out[1]` maximum addressable height of screen/plotter in rasters/steps assuming a 0 start point; for example, a resolution of 480 implies an addressable area of 0-479, so `work_out[1]=479`

`work_out[2]` device coordinate units flag:

- 0 device capable of producing precisely scaled image; typically plotters and printers
- 1 device not capable of precisely scaled image (CRTs)

`work_out[3]` width of one pixel (plotter step) in micrometers

`work_out[4]` height of one pixel (plotter step) in micrometers

`work_out[5]` number of character heights available; 0 = continuous scaling

`work_out[6]` number of line types available; 0 = device is not capable of graphics

`work_out[7]` number of line widths available

`work_out[8]` number of marker types available

`work_out[9]` number of marker sizes available; 0 = continuous scaling

`work_out[10]` number of graphics text fonts available

`work_out[11]` number of patterns available

work_out[12] number of hatch styles available

work_out[13] number of colors simultaneously displayable on this device; at least 2, even for a monochrome device

work_out[14] number of Generalized Drawing Primitives (GDPs)

work_out[15] to work_out[24] list of available GDPs (up to 10 allowed); the list can be specified in any order:

- 1 GDP does not exist
- 1 bar
- 2 arc
- 3 pie slice
- 4 circle
- 5 elliptical arc
- 6 elliptical pie slice
- 7 ellipse

work_out[25] to work_out[34] attribute set associated with each GDP:

- 1 GDP does not exist
- 0 polyline
- 1 polymarker
- 2 text
- 3 fill area
- 4 none
- 5 other

work_out[35]	color capability flag: 0 no 1 yes
work_out[36]	text rotation capability flag: 0 no 1 yes
work_out[37]	fill area capability flag: 0 no 1 yes
work_out[38]	pixel operation capability flag; indicates whether the device is capable of cell array primitive operations: 0 no 1 yes
work_out[39]	number of available colors; total number of colors in color palette (may not be displayable all at one time): 0 continuous device 2 monochrome (black and white) >2 number of colors available
work_out[40]	locator capability flag: 0 no 1 input device is a keyboard or a mouse with one button >1 number of buttons on the input device
work_out[41]	valuator capability flag: 0 no 1 yes

- work_out[42] choice capability flag; the value returned is the number of Choice indicators available; for example, if 5 function keys are used as Choice devices, the value returned would be 5
- work_out[43] string input capability flag:
 - 0 no
 - 1 yes
- work_out[44] workstation type:
 - 0 output only
 - 1 inputonly
 - 2 input/output
 - 3 device-independent segment storage
 - 4 metafile output
 - 5 other
- work_out[45] device type:
 - 0 CRT
 - 1 plotter
 - 2 printer
 - 3 camera/film recorder
 - 4 metafile output
 - 5 other
- work_out[46] number of writing modes available
- work_out[47] highest level of input mode available:
 - 0 none
 - 1 request
 - 2 sample

work_out[48]	text alignment capability flag: 0 no 1 yes
work_out[49]	inking capability flag as output echo device: 0 no 1 yes
work_out[50]	rubberbanding capability flag as output echo device: 0 no rubberband capability 1. capable of rubberband lines 2 capable of rubberband lines and rectangles
work_out[51]	maximum addressable VDC space coordinate on <i>x</i> axis; this value is filled in based on the coordinate transformation mode selected in <i>work_in</i> [0]
work_out[52]	maximum addressable VDC space coordinate on <i>y</i> axis; this value is filled in based on the coordinate transformation mode selected in <i>work_in</i> [0]
work_out[53] to work_out[57]	version of the driver (this is an ADE character string that represents the version of the driver in the following form: <i>vv.ll</i> where <i>vv</i> is the actual version and <i>ll</i> is the level)
work_out[58] and work_out[59]	reserved for future use
work_out[60]	minimum graphics text character height in VDC units
work_out[61]	maximum graphics text character height in VDC units
work_out[62]	minimum line width in VDC units

SCO CGI Programmer's Guide

<code>work_out[63]</code>	maximum line width in VDC units
<code>work_out[64]</code>	minimum marker height in VDC units
<code>work_out[65]</code>	maximum marker height in VDC units

Value Returned

0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Clear Workstation
Close Workstation
Update Workstation

Description

The Open Workstation function initializes a workstation (a graphics device). It sets all defaults and returns device information. Both the alpha and graphics display surfaces are cleared by this function.

The following four transformation modes are available.

- Mode 0** This mode maps VDC space to full extent of each axis. This mode does not preserve aspect ratio. The picture will completely fill the screen.
- Mode 1** This mode maps VDC space to the full extent of the longest display surface axis only. It maps a subset of VDC space to the shorter axis. This mode preserves unity aspect ratio. Using this technique and the appropriate scaling factor, results in a picture with the same aspect ratio.
- Mode 2** In this mode all coordinates are specified in physical device-dependent units. This mode enables the application to address physical device dependent coordinates. The physical VDC extent is device dependent. This mode is useful for raster-oriented applications that need to access specific physical pixels.

Mode3 This mode maps VDC space to the full extent of the shortest display surface axis. It maps a subset of VDC space to the longest display surface axis in such a way as to preserve a unity aspect ratio.

Transformation modes are discussed in detail in Chapter 2, **The Graphics Model**, in this manual.

The input parameters *work_in*[11] to *work_in*[18] are an ADE form that is used to determine which driver to dynamically bring into memory, but is not used by the driver itself. Any unique string of up to eight characters can be used for the logical device name.

The output parameter *work_out*[13] is the number of colors that can be displayed on the device simultaneously when in Graphics Mode. The number of colors in Cursor Addressing Mode may be different.

3.62 Output Alpha Text

Class

Output Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>string</code>	text string

Outputs

<code>x_out</code>	<i>x</i> coordinate of text position (in VDC units) after the text string has been output (this is the same value that is returned if the Inquire Alpha Text Position function were invoked)
<code>y_out</code>	<i>y</i> coordinate of text position (in VDC units) after the text string has been output (this is the same value that is returned if the Inquire Alpha Text Position function were invoked)

Value Returned

0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Inquire Alpha Text Capabilities
 Inquire Alpha Text Cell Location
 Inquire Alpha Text Font Capability
 Inquire Alpha Text Position
 Inquire Alpha Text String Length
 Set Alpha Text Color Index
 Set Alpha Text Font And Size
 Set Alpha Text Line Spacing
 Set Alpha Text Overstrike Mode
 Set Alpha Text Pass Through Mode
 Set Alpha Text Position
 Set Alpha Text Quality
 Set Alpha Text Subscript/Superscript Mode
 Set Alpha Text Underline Mode

Description

The Output Alpha Text function outputs text at the current alpha text position, honoring all current alpha text attributes (for example, subscripting, underline, and alpha text font). The cursor is positioned at the end of the string. The alpha text position is updated appropriately after outputting the text string. Receipt of the ASCII character <CR> (carriage return) causes the alpha text position to be set to the beginning of the line ($x=0$). Receipt of a <LF> (line feed) causes the alpha text position to be advanced by the current line spacing ($y=y - \text{line spacing}$). All other control characters/nonprintable characters (ASCII characters 0-31) are not output. Attempting to display characters past the x or y maximum of the display surface produces device-dependent results.

Alpha text is useful for outputting word processor quality text display on CRTs and printers, and is displayed to the best resolution and accuracy of the hardware.

3.63 Output Arc

Class

Output Functions

Inputs

dev_handle	CGIhandle for a currently open workstation
x	x coordinate of center point of arc in VDC units
y	y coordinate of centerpoint of arc in VDC units
radius	radius
begang	start angle in tenths of degrees (0-3600)
endang	end angle in tenths of degrees (0-3600)

Outputs

None

Value Returned

0	no error
-1	error occurred; actual error can be retrieved by invoking <code>InquireCGIError</code>

Related Functions

Inquire Background Mode
Inquire Clip Rectangle
Inquire Drawing Bitmap
Inquire Line Representation
Select Drawing Bitmap
Set Background Color Index
Set Background Mode
Set Clip Rectangle
Set Line Color Index
Set Line Cross Section
Set Line Type
Set Line Width
Set Line Representation
Set User Line Type
Set Writing Mode

Description

The Output Arc function draws arcs using current line attributes. Arcs are defined by the center point and two end points of the arc. The radius is assumed to be measured along the x (horizontal) axis. All angle specifications assume that 0° is at 3 o'clock, with values increasing in the counterclockwise direction.

Arcs and pie slices are always drawn in a counterclockwise direction. The start angle does not need to be larger than the ending angle. If a start angle of 40° and an ending angle of 15° are given, an arc would be drawn counterclockwise from the 40° angle to the 15° angle.

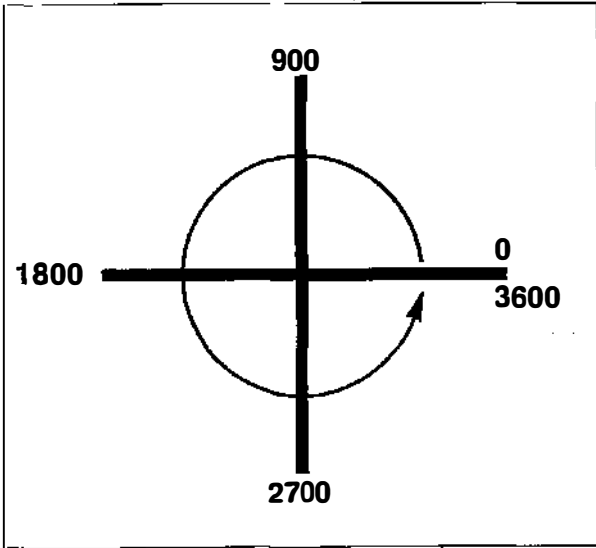


Figure 3-1. *Angle Specification*

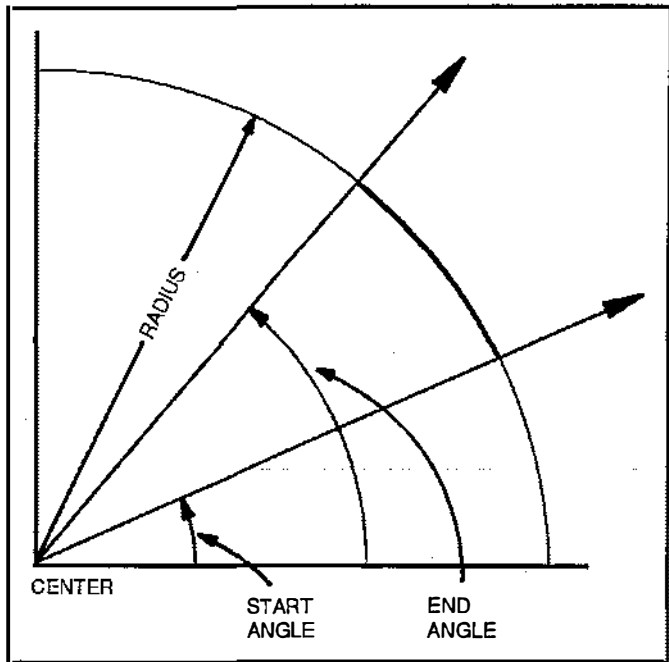


Figure 3-2. Arc

3.64 Output Bar

Class

Output Functions

Inputs

<code>dev_handle</code>	CGIhandle for a currently open workstation
<code>xy[0]</code>	<i>x</i> coordinate of lower left-hand corner of bar in VDCunits
<code>xy[1]</code>	<i>y</i> coordinate of lower left-hand corner of bar in VDCunits
<code>xy[2]</code>	<i>x</i> coordinate of upper right-hand corner of bar in VDCunits
<code>xy[3]</code>	<i>y</i> coordinate of upper right-hand corner of bar in VDCunits

Outputs

None

Value Returned

0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Inquire Background Mode
Inquire Clip Rectangle
Inquire Drawing Bitmap
Inquire Fill Area Representation
Select Drawing Bitmap
Set Background Color Index
Set Background Mode
Set Clip Rectangle
Set Fill Area Representation
Set Fill Color Index
Set Fill Interior Style
Set Fill Style Index
Set Writing Mode

Description

The Output Bar function draws a rectangular area using current filled area attributes of interior style, fill style and color.

3.65 OutputByte Pixel Array

Class

Output Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>origin[0]</code>	<i>x</i> coordinate, in VDC units, of one corner of destination rectangle
<code>origin[1]</code>	<i>y</i> coordinate, in VDC units, of one corner of destination rectangle
<code>width</code>	width (in pixels) of the source pixel array
<code>height</code>	height (in pixels) of the source pixel array
<code>valid_width[0]</code>	first valid column index
<code>valid_width[1]</code>	last valid column index
<code>valid_height[0]</code>	first valid row index
<code>valid_height[1]</code>	last valid row index
<code>pixels[]</code>	array (<code>height*width</code>) holding byte pixels to output to workstation

Outputs

None

Value Returned

≥ 0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Inquire Byte Pixel Array
 Inquire Clip Rectangle
 Inquire Color Representation
 Inquire Integer Pixel Array
 Output Integer Pixel Array
 Set Background Mode
 Set Clip Rectangle
 Set Color Representation
 Set Color Table
 Set Writing Mode

Description

The Output Byte Pixel Array function assigns a rectangular array of device independent color indices to a rectangular array of pixels on the specified output device using the current writing mode and honoring the current clipping rectangle. The only difference between the Output Byte Pixel Array function and the Output Integer Pixel Array function is that each pixel array element of Output Byte Pixel Array can represent color indices between 0 and 255 while each element of Output Integer Pixel Array can represent color indices between 0 and 65535.

The pixel array is passed as a vector where each row of the pixel array follows the previous row in the vector. The data are passed in the array from lowest row to highest row and from lowest column to highest column within each row. A pixel array with n rows and m columns is represented as follows:

Array Element	Description
0	row1, col1
1	row1, col2
.	.
.	.
$m - 1$	row1, colm
m	row2, col1
$m + 1$	row2, col2
.	.
.	.
$2 * 2^p * m - 1$	row2, colm
.	.
.	.
$n * 2^p * m - 1$	rown, colm

width is a signed integer that defines how many pixels are in each row. If *width* is greater than 0 then the row extends toward increasing *x* from the specified origin. If *width* is less than 0, then the row extends toward decreasing *x* from the specified origin.

height is a signed integer that defines how many rows of pixels there are. If *height* is greater than 0 then the rows extend towards increasing *y* from the specified origin. If *height* is less than 0, then the rows extend towards decreasing *y* from the specified origin.

If *width*height* is zero (0), no pixels are drawn.

valid_width and *valid_height* specify the first and last values of $\text{abs}(\text{width}) - 1$ and $\text{abs}(\text{height}) - 1$ that contain defined color indices (pixels). If the entire array contains defined color indices, then

valid_width = (0, $\text{abs}(\text{width}) - 1$) and *valid_height* = (0, $\text{abs}(\text{height}) - 1$).
Valid_width and *valid_height* provide a means to specify a subregion of the pixel array to be displayed.

Pixel arrays are limited by the physical limits of the selected bitmap.

Output pixel arrays are affected by the current Writing Mode, Background Mode, and Clip Rectangle.

For the simplest applications of this function, the *valid_width* parameter is (0, $\text{abs}(\text{width}) - 1$) and the *valid_height* parameter is (0, $\text{abs}(\text{height}) - 1$). For example, if a pixel array is defined such that its *width* is 64 and its *height* is 128, the application could display a subregion with a *width* of 32, *height* of 64, and the lower left corner of the subregion at row 16 and column 16 from the lower left corner of the rectangular area as in the following pseudocode example:

```
char pixels (8192)
int origin /100,60/
int valid_width /15,47/
int valid_height /15,79/
int width /64/
int height /128/
```

```
BYTE_PIXEL_ARRAY(dev_handle, origin, width, height, valid_width,
valid_height, pixels)
```


3.66 OutputCellArray**Class****Output Functions****Inputs**

<code>dev_handle</code>	CGIhandle for a currently open workstation
<code>xy[0]</code>	x coordinate of lower left corner in VDC units
<code>xy[1]</code>	y coordinate of lower left corner in VDC units
<code>xy[2]</code>	x coordinate of upper right corner in VDC units
<code>xy[3]</code>	y coordinate of upper right corner in VDC units
<code>row_length</code>	length of each row in color index array
<code>el_per_row</code>	number of elements used in each row of color index array
<code>num_rows</code>	number of rows in color index array
<code>wr_mode</code>	pixel operation to be performed (see Set Writing Mode for list of operations)
<code>colors</code>	color index array (stored one row at time)

Outputs

None

Value Returned

0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

- Inquire Clip Rectangle
- Inquire Color Representation
- Inquire Drawing Bitmap
- Select Drawing Bitmap
- Set Clip Rectangle
- Set Color Representation
- Set Color Table
- Set Writing Mode

Description

The Output Cell Array function outputs a rectangular grid of color blocks to the output device.

Each row in the color index array is expanded evenly to fill the entire width of the rectangle specified by pixel replication. Each row in the color index array is also replicated the appropriate number of times to fill the entire height of the rectangular area evenly with the color pattern, starting from the top of the rectangular area and filling downward.

For example, if there are two rows with three elements per row, the vertical dimension of the rectangle is divided into two equal parts, and the horizontal dimension is divided into three equal parts. The rectangle is filled in the upper left-hand corner of the area with the color index specified in the first element of the color index array. If the device can't do cell arrays, the area is outlined in the current line color.

The input array to the cell array function is a list of color indices. Each of these indices indicates the color to use to fill one of the cells of the cell array. The *xy* array defines the corners of the rectangle. *el_per_row* and *num_rows* define how many sections (cells) to divide the rectangle into. For example, to create the cell array shown in Figure 3-3 the following values would be entered:

```
xy[0] = 24
xy[1] = 300
xy[2] = 50
xy[3] = 500
row_length = 3
el_per_row = 3
num_rows = 2
wr_mode = 4
colors[0] = 2
colors[1] = 4
colors[2] = 5
colors[3] = 3
colors[4] = 1
colors[5] = 4
```

We could also have used the following set of values:

```
xy[0] = 24
xy[1] = 300
xy[2] = 50
xy[3] = 500
row_length = 5
el_per_row = 3
num_rows = 2
wr_mode = 4
colors[0] = 2
colors[1] = 4
colors[2] = 5
colors[3] = data value not used
colors[4] = data value not used
colors[5] = 3
colors[6] = 1
colors[7] = 4
colors[8] = data value not used
colors[9] = data value not used
```

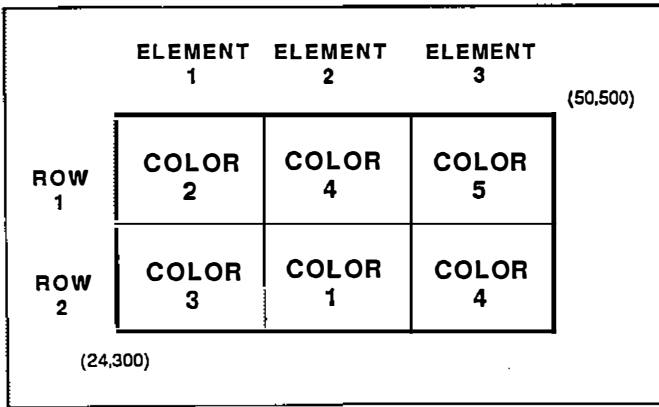


Figure 3-3. Cell Array

In the second example, the *row_length* is set to 5 and the colors array has been increased. Because the *elements_per_row* value is 3, we only use the first three of the five elements per row. This mechanism is useful when the user only wants to use a portion of a large array of data.

The *colors* values correspond to entries from the currently defined color table.

In this example, we used a writing mode value of four. However, any of the sixteen writing mode values could have been used.

Each *pixel* within a cell is displayed with the indicated color. By specifying an array that is the same size resolution as the display device and indicating the colors of the display, the user could set the color of each individual pixel on the screen using the current color graphics card.

See the Open Workstation parameter *work_out*[38] to determine if the device represented by *dev_handle* is capable of cell array operations.

3.67 Output CGIError

Class

Output Functions

Inputs

string text string to prepend to the error message

Outputs

None

Value Returned

None

Related Functions

Inquire CGIError

Description

The Output CGIError function displays the error message associated with the last encountered CGI error on *stderr*.

Nothing is displayed if no error has yet occurred.

3.68 Output Circle

Class

Output Functions

Inputs

dev_handle	CGI handle for a currently open workstation
x	x coordinate of center point of circle in VDC units
y	y coordinate of center point of circle in VDC units
radius	radius

Outputs


None

Value Returned



0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

- Inquire Background Mode
- Inquire Clip Rectangle
- Inquire Drawing Bitmap
- Set Background Color Index
- Set Background Mode
- Set Clip Rectangle
- Set Fill Area Representation
- Set Fill Color Index
- Set Fill Interior Style
- Set Fill Style Index
- Set Writing Mode

Description

The Output Circle function draws a circle specified by a center point and radius. The radius is assumed to be measured along the x (horizontal) axis. Because circles are a special type of filled area, they are affected by filled area attributes including interior style, fill style and color.



3.69 Output Cursor Addressable Text

Class

Output Functions

Inputs

dev_handle	CGIhandle for a currently open workstation
string	text string

Outputs

None

Value Returned

0	no error
-1	error occurred; actual error can be retrieved by invoking Inquire CGIError

Related Functions

- Cursor Down
- Cursor Home
- Cursor Left
- Cursor Right
- Cursor Up
- Direct Cursor Address
- Enter Cursor Addressing Mode
- Inquire Current Cursor Text Address
- Reverse Video Off
- Reverse Video On
- Set Cursor Text Attributes
- Set Cursor Text Color Index

Description

For CRTs, the Output Cursor Addressable Text function outputs text at the current cursor position, honoring all the cursor text attributes (such as color and reverse video). New text replaces (overwrites) old text at the same location.

Only text that will fit onto the line of the cursor will be displayed; it will not wrap onto the next line at the left edge.

When the cursor is on the bottom line of the screen and characters are subsequently sent to the screen, any characters that exceed the right margin will wrap around to the left margin. Any characters already on the last line will be overwritten.

Note

You must be in Cursor Addressing Mode to output cursor addressable text.

3.70 Output Ellipse

Class

Output Functions

Inputs

dev_handle	CGI handle for a currently open workstation
x	x coordinate of center of ellipse in VDC units
y	y coordinate of center of ellipse in VDC units
x_radius	radius of ellipse along the x axis in VDC units
y_radius	radius of ellipse along the y axis in VDC units

Outputs

None

Value Returned

≥0	no error
-1	error occurred; actual error can be retrieved by invoking Inquire CGIError

Related Functions

- Inquire Background Mode
- Inquire Clip Rectangle
- Inquire Drawing Bitmap
- Select Drawing Bitmap
- Set Background Color Index
- Set Background Mode
- Set Clip Rectangle
- Set Fill Area Representation
- Set Fill Color Index
- Set Fill Interior Style
- Set Fill Style Index
- Set Writing Mode

Description

An ellipse is drawn with center point at x,y and with the specified x_radius and y_radius (see Figure 3-4). The x and y radii are along their respective axes and cannot be rotated.

All fill area attributes apply to this function. Portions of the ellipse that are outside the clip rectangle are not displayed.

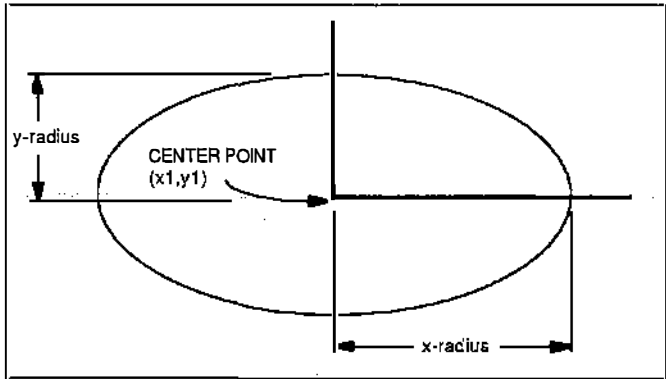


Figure 3-4. *Ellipse*

3.71 OutputEllipticalArc

Class

Output Functions

Inputs

dev_handle	CGI handle for a currently open workstation
x	x coordinate of center of ellipse in VDC units
y	y coordinate of center of ellipse in VDC units
x_radius	radius of ellipse along the x axis in VDC units
y_radius	radius of ellipse along the y axis in VDC units
start_angle	start angle of arc in tenths of degrees (0-3600)
end_angle	end angle of arc in tenths of degrees (0-3600)

Outputs

None

Value Returned

≥0	no error
-1	error occurred; actual error can be retrieved by invoking Inquire CGIError

Related Functions

Inquire Background Mode
 Inquire Clip Rectangle
 Inquire Drawing Bitmap
 Inquire Line Representation
 Select Drawing Bitmap
 Set Background Color Index
 Set Background Mode
 Set Clip Rectangle
 Set Line Color Index
 Set Line Cross Section
 Set Line Type
 Set Line Width
 Set Line Representation
 Set User Line Type
 Set Writing Mode

Description

An elliptical arc is drawn with center point at x,y and with the specified x_radius , y_radius , $start_angle$, and end_angle (see Figure 3-5). The x and y radii are along their respective axes and cannot be rotated. The start and end angles are specified as if the figure being drawn were a circle. These angles are then skewed to the same degree that the ellipse is a skewed circle. For an ellipse with a large x_radius and small y_radius , 45° is closer to the x axis than for a circle, where a 45° line is equidistant from both the x and y axes.

All polyline attributes apply to this function. Portions of the arc that are outside the clip rectangle are not displayed.

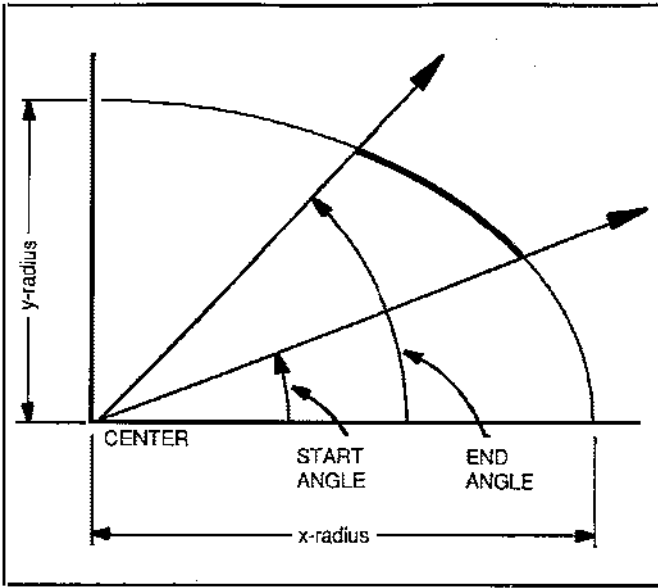


Figure 3-5. *Elliptical Arc*

3.72 Output Elliptical Pie Slice**Class**

Output Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>x</code>	<i>x</i> coordinate of center of ellipse in VDC units
<code>y</code>	<i>y</i> coordinate of center of ellipse in VDC units
<code>x_radius</code>	radius of ellipse along the <i>x</i> axis in VDC units
<code>y_radius</code>	radius of ellipse along the <i>y</i> axis in VDC units
<code>start_angle</code>	start angle of pie slice in tenths of degrees (0-3600)
<code>end_angle</code>	end angle of pie slice in tenths of degrees (0-3600)

Outputs

None

Value Returned

≥ 0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Inquire Background Mode
Inquire Clip Rectangle
Inquire Drawing Bitmap
Select Drawing Bitmap
Set Background Color Index
Set Background Mode
Set Clip Rectangle
Set Fill Color Index
Set Fill Interior Style
Set Fill Style Index
Set Writing Mode

Description

An elliptical pie slice is drawn with center point at x,y and with the specified x_radius , y_radius , $start_angle$, and end_angle . The x and y radii are along their respective axes and cannot be rotated. The start and end angles are specified as if the figure being drawn were a circle. These angles are then skewed to the same degree that the ellipse is a skewed circle. For an ellipse with a large x_radius and small y_radius , 45° is closer to the x axis than for a circle, where a 45° line is equidistant from both the x and y axes.

All fill area attributes apply to this function. Portions of the elliptical pie slice that are outside the clip rectangle are not displayed.

3.73 Output Filled Area**Class****Output Functions****Inputs**

dev_handle	CGI handle for a currently open workstation
count	number of vertices in polyline
xy[0]	x coordinate of first point of polyline in VDC units
xy[1]	y coordinate of first point of polyline in VDC units
xy[2]	x coordinate of second point of polyline in VDC units
xy[3]	y coordinate of second point of polyline in VDC units
	.
	.
	.
xy[2*count-2]	x coordinate of last point of polyline in VDC units
xy[2*count-1]	y coordinate of last point of polyline in VDC units

Outputs

None

Value Returned

0	no error
-1	error occurred; actual error can be retrieved by invoking Inquire CGIError

Related Functions

- Inquire Background Mode
- Inquire Clip Rectangle
- Inquire Drawing Bitmap
- Inquire Fill Area Representation
- Select Drawing Bitmap
- Set Background Color Index
- Set Background Mode
- Set Clip Rectangle
- Set Fill Area Representation
- Set Fill Color Index
- Set Fill Interior Style
- Set Fill Style Index
- Set Writing Mode

Description

The Output Filled Area function outputs a filled area to the device using current fill area attributes of fill color index, fill interior style index, fill style index. Hollow filled areas are outlined with a solid border using the current fill color. Solid, hatch, and pattern-filled areas are not outlined. See Figure 3-6.

Portions of the filled area that are outside the clip rectangle are not displayed.

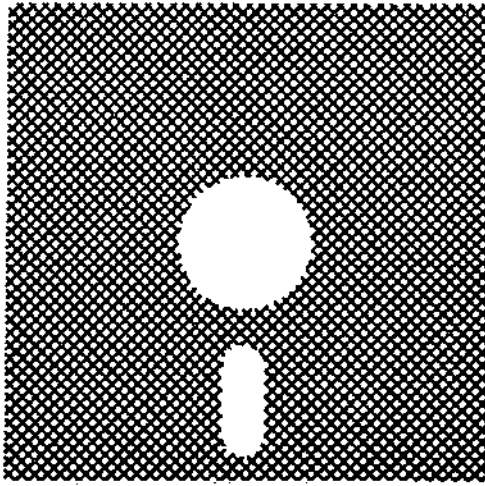


Figure 3-6. *Filled Area*

3.74 OutputGraphics Text

Class

Output Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>x</code>	<code>x</code> coordinate of alignment point of text in VDC units
<code>y</code>	<code>y</code> coordinate of alignment point of text in VDC units
<code>string</code>	text string; characters must be printable characters, ASCII space and above

Outputs

None

Value Returned

0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Inquire Background Mode
Inquire Clip Rectangle
Inquire Drawing Bitmap
Inquire Graphics Text Extent
Inquire Graphics Text Representation
Select Drawing Bitmap
Set Background Color Index
Set Background Mode
Set Clip Rectangle
Set Graphics Text Alignment
Set Graphics Text Character Height
Set Graphics Text Color Index
Set Graphics Text Font
Set Graphics Text Representation
Set Graphics Text String Baseline Rotation
Set Writing Mode

Description

The Output Graphics Text function outputs graphics text with current attributes to the device. The x, y location is the starting position of the text alignment point. The default alignment is such that the x, y location corresponds to the lower left corner of the character. Any text that is outside of the clip rectangle is not displayed.

3.75 OutputIntegerPixel Array

Class

Output Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>origin[0]</code>	<i>x</i> coordinate, in VDC units, of one corner of destination rectangle
<code>origin[1]</code>	<i>y</i> coordinate, in VDC units, of one corner of destination rectangle
<code>width</code>	width (in pixels) of the source pixel array
<code>height</code>	height (in pixels) of the source pixel array
<code>valid_width[0]</code>	first valid column index
<code>valid_width[1]</code>	last valid column index
<code>valid_height[0]</code>	first valid row index
<code>valid_height[1]</code>	last valid row index
<code>pixels[]</code>	array (<code>height*width</code>) holding byte pixels to output to workstation

Outputs

None

Value Returned

<code>≥0</code>	no error
<code>-1</code>	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Inquire Byte Pixel Array
Inquire Clip Rectangle
Inquire Color Representation
Inquire Integer Pixel Array
Output Byte Pixel Array
Set Background Mode
Set Clip Rectangle
Set Color Representation
Set Color Table
Set Writing Mode

Description

The Output Integer Pixel Array function assigns a rectangular array of device independent color indices to a rectangular array of pixels on the specified output device using the current writing mode and honoring the current clipping rectangle.

The only difference between the Byte Pixel Array function and the Integer Pixel Array function is that each pixel array element of Byte Pixel Array can represent color indices between 0 and 255 and Integer Pixel Array can represent color indices between 0 and 65535.

Pixel arrays are limited by the physical limits of the selected bitmap.

Output pixel arrays are affected by the current writing mode, background mode, and clip rectangle.

3.76 Output Pie Slice

Class

Output Functions

Inputs

dev_handle	CGI handle for a currently open workstation
x	x coordinate of center point of arc in VDC units
y	y coordinate of center point of arc in VDC units
radius	radius in VDC units
begang	start angle in tenths of degrees; 0-3600
endang	end angle in tenths of degrees; 0-3600

Outputs

None

Value Returned

0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

`Inquire Background Mode`
`Inquire Clip Rectangle`
`Inquire Drawing Bitmap`
`Inquire Fill Area Representation`
`Select Drawing Bitmap`
`Set Background Color Index`
`Set Background Mode`
`Set Clip Rectangle`
`Set Fill Area Representation`
`Set Fill Color Index`
`Set Fill Interior Style`
`Set Fill Style Index`
`Set Writing Mode`

Description

The Output Pie Slice function draws pie slices by specifying the center point and two points on the arc. The radius is assumed to be measured along the x (horizontal) axis. Because pie slices are a special type of filled area, they are affected by filled area attributes including interior style, fill style, and color.

All angle specifications assume that 0° is at 3 o'clock, with values increasing in the counterclockwise direction. See Figure 3-1, *Angle Specification* in the OutputArc function.

Portions of the pie slice that are outside the clip rectangle are not displayed.

3.77 OutputPolyline

Class

Output Functions

Inputs

dev_handle	CGI handle for a currently open workstation
n	number of vertices (x,y pairs) in polyline
xy[0]	x coordinate of first point of polyline in VDC units
xy[1]	y coordinate of first point of polyline in VDC units
xy[2]	x coordinate of second point of polyline in VDC units
xy[3]	y coordinate of second point of polyline in VDC units
	.
	.
	.
xy[2*n-2]	x coordinate of last point of polyline in VDC units
xy[2*n-1]	y coordinate of last point of polyline in VDC units

Outputs

None

Value Returned

0	no error
-1	error occurred; actual error can be retrieved by invoking <code>InquireCGIError</code>

Related Functions

Inquire Background Mode
Inquire Clip Rectangle
Inquire Drawing Bitmap
Inquire Line Representation
Select Drawing Bitmap
Set Background Color Index
Set Background Mode
Set Clip Rectangle
Set Line Color Index
Set Line Cross Section
Set Line Type
Set Line Width
Set Line Representation
Set User Line Type
Set Writing Mode

Description

The Output Polyline function outputs a polyline with the current polyline attributes of line style, width and color. It moves to the first point and draws a line between subsequent points. Line type is continued between subsequent points of a polyline. Line type is not continued between two separate polyline functions.

Portions of the polyline that are outside the clip rectangle are not displayed.

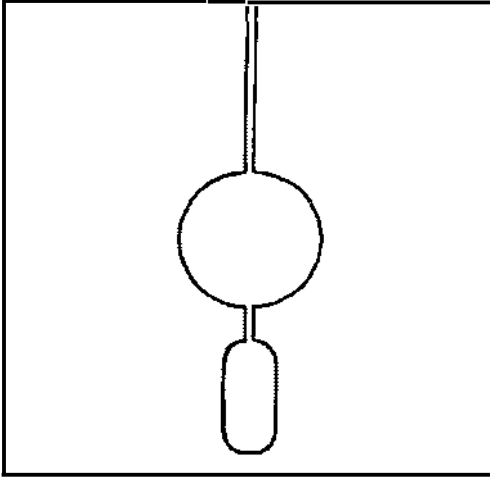


Figure 3-7. *Polyline*

3.78 OutputPolymarker**Class****Output Functions****Inputs**

dev_handle	CGI handle for a currently open workstation
n	number of markers
xy[0]	x coordinate of first marker in VDC units
xy[1]	y coordinate of first marker in VDC units
xy[2]	x coordinate of second marker in VDC units
xy[3]	y coordinate of second marker in VDC units
...	...
xy[2*n-2]	x coordinate of last marker in VDC units
xy[2*n-1]	y coordinate of last marker in VDC units

Outputs

None

Value Returned

0	no error
-1	error occurred; actual error can be retrieved by invoking Inquire CGIError

Related Functions

- Inquire Background Mode
- Inquire Clip Rectangle
- Inquire Drawing Bitmap
- Inquire Marker Representation
- Select Drawing Bitmap
- Set Background Color Index
- Set Background Mode
- Set Clip Rectangle
- Set Marker Color Index
- Set Marker Height
- Set Marker Representation
- Set Marker Type
- Set Writing Mode

Description

The Output Polymarker function outputs markers with the current polymarker attributes of scale, type and color. At least six marker types are provided as specified in the Set Marker Type function.

Portions of the polymarker that are outside the clip rectangle are not displayed.

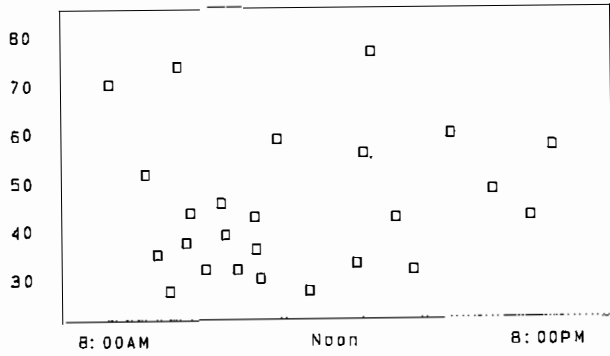


Figure 3-8. *Polymarker*

3.79 Read Curs or Keys

Class

Input Functions

Inputs

`dev_handle` CGI handle for a currently open workstation

`input_mode` requested input mode:

- 1 Request
- 2 Sample

Outputs

`direction` the returned value determines the definition of this parameter as follows:

- 1 cursor key, direction indicated
- 2 string key, `direction = 0`
- 3 choice key, choice index
- 4 special key, special key index
- 5 no cursor movement

For cursor key, this parameter is defined as follows:

- 1 down and left
- 2 down
- 3 down and right
- 4 left
- 6 right
- 7 up and left
- 8 up
- 9 up and right

For special keys, this parameter is defined as follows:

- 1 home / beginning (if different from up and left)
- 2 end (if different from down and left)
- 3 page up (if different from up and right)
- 4 page down (if different from down and right)
- 5 insert char

- 6 delete char
- 7 insert line
- 8 delete line
- 9 erase to end of line
- 10 erase to end of page
- 11 clear page
- 12 cancel
- 13 next
- 14 previous
- 15 help
- 16 print (print screen)
- 17 enter (if different from return)

key the ADE character value (0-127) of the key pressed if ASCII key pressed; otherwise undefined

Value Returned

- 0 no key stroke occurred or an unknown key was pressed
- 1 error occurred; actual error can be retrieved by invoking `Inquire CGIError`
- 1 cursor key pressed
- 2 string/ASCII key pressed
- 3 choice/function key pressed
- 4 special key pressed
- 5 no cursor movement

Related Functions

Request Choice
Request Locator
Request String
Sample Choice
Sample Locator
Sample String

Description

The Read Cursor Keys function determines the type and value of a key press on a keyboard. For devices similar to the IBM PC keyboard there are a number of possible key strokes possible. One of the following types of keys may be pressed:

- an ASCII key
- a cursor direction key
- a choice (function) key
- a special key.

Note

This function is upwardly compatible with the GSS-DRIVERS 1.0, except that the function now returns positive values other than 0. Applications that only tested for the error condition value returned less than zero, will have no problems with the additional functionality.

3.80 Remove CGI**Class**

ControlFunctions

Inputs

None

Outputs

None

Value Returned

≥0 no error

-1 error occurred; actual error can be retrieved by
invoking Inquire CGIError**Related Functions**Close File
Close Workstation
LoadCGI**Description**

The Remove CGI function cleans up the area devoted to the CGI and its drivers so that the area can be used for other purposes.

Note

No file closing or disconnecting is performed; data left in buffers from any open workstations may be lost.

This function is valid only for DOS. In the interest of code portability, this function is present in the XENIX libraries and will return -1.

3.81 Remove Graphics Input Cursor

Class

Control Functions

Inputs

dev_handle CGI handle for a currently open workstation

Outputs

None

Value Returned

0 no error

-1 error occurred; actual error can be retrieved by
invoking Inquire CGIError

Related Functions

Display Graphics Input Cursor
Request Locator
Sample Locator

Description

The Remove Graphics Input Cursor function removes the graphics input cursor from its current location on the screen. It does not need to be referenced when using Request Locator.

3.82 Request Choice

Class

Input Functions

Inputs

<code>dev_handle</code>	CGIhandle for a currently open workstation
<code>ch_in</code>	initial choice number

Outputs

<code>ch_out</code>	choicenumber
---------------------	--------------

Value Returned

≥ 0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Open Workstation
Sample Choice

Description

The Request Choice function activates the choice device associated with a specified workstation. The operator must make a selection for the function to terminate.

The initial choice number parameter, `ch_in`, is the value returned as the selected choice value if a non-choice device button/key is pressed. The initial choice number must be a valid value for it to be returned when the user does not press a valid choice input. Keyboard function keys are examples of choices.

See the Open Workstation parameter `work_out[42]` to determine if the device represented by `dev_handle` is capable of Choice Input.

3.83 Request Locator

Class

Input Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open input-capable workstation
<code>xy_in[0]</code>	initial <i>x</i> coordinate of locator in VDC units
<code>xy_in[1]</code>	initial <i>y</i> coordinate of locator in VDC units
<code>ink</code>	inking status: 0 off 1 on; a line is drawn between the initial locator position and the final locator position, honoring the current line attributes, such as color and line type
<code>rubberband</code>	rubberbanding status: 0 rubberbanding off 1 rubberband line; a line is drawn between the initial locator position and the current position of the locator device as it is moved; the line changes dynamically as the input device changes position; when the locator event is complete, the last rubberband line is removed from the display surface 2 rubberband rectangle; a rectangle is drawn using the initial locator position as one corner and the current position of the locator device as the opposite corner; the rectangle changes dynamically as the input device changes position; when the locator event is complete, the last rubberband rectangle is removed from the display surface

`echo_handle` the *dev_handle* of the device where the echoed feedback from the input operation will be displayed

Outputs

<code>xy_out[0]</code>	final <i>x</i> coordinate of locator in VDC units
<code>xy_out[1]</code>	final <i>y</i> coordinate of locator in VDC units
<code>terminator</code>	locator terminator

Value Returned

≥ 0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

- Create Cursor
- Delete Cursor Description
- Inquire Cursor Description
- Inquire Graphics Input Cursor
- Inquire Input Extent
- Inquire Line Representation
- Open Workstation
- Open Workstation
- Sample Locator
- Select Graphics Input Cursor
- Set Input Extent
- Set Line Color Index
- Set Line Cross Section
- Set Line Type
- Set Line Width
- Set Line Representation
- Set User Line Type
- Set Writing Mode

Description

The Request Locator function tracks the current position of the input device by displaying the graphics cursor on the output echo device until the input is terminated. This function returns the final location of the input device and the key pressed to terminate the input.

If an invalid inking status is specified, inking is turned off.

Rubberbanding honors current line attributes, such as color and line style. If an invalid rubberband status is specified, rubberbanding is turned off.

For keyboard terminated locator input, the output parameter *terminator* is the character value of the key struck to terminate input. For non-keyboard terminated input (such as with a tablet or a mouse) this parameter is device-dependent. Refer to the *Device Driver Supplement to the SCO CGI Programmer's Guide* for specific information.

See the Open Workstation parameter *work_out*[40] to determine whether or not the device represented by *dev_handle* has Locator Input capabilities. See the Open Workstation parameters *work_out*[49] and *work_out*[50] to determine whether or not the device represented by *dev_handle* has inking and rubberbanding capabilities.

3.84 RequestString**Class****Input Functions****Inputs**

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>max_length</code>	maximum string length
<code>echo_mode</code>	<p>0 don't echo input characters</p> <p>1 echo input characters</p>
<code>echo_xy[0]</code>	x coordinate of echo position in VDC units
<code>echo_xy[1]</code>	y coordinate of echo position in VDC units

Outputs

<code>string</code>	null-terminated text string
---------------------	-----------------------------

Value Returned

≥ 0	length of string
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Open Workstation
 Sample String
 Set Line Edit Characters

Description

The Request String function activates the keyboard. Any characters up to (but not including) the carriage return or line feed terminator are returned. Line editing characters have their normal effect and can be used if errors are made. The maximum string length must be greater than or equal to one. This function terminates when the *max_length* number of characters or a line terminator has been entered.

Because strings are null-terminated, the array and *max_length* must be one greater than the required value to accommodate the null at the end of the string.

See the Open Workstation parameter *work_out*[43] to determine if the device referred to by *dev_handle* is capable of String Input.

3.85 Request Valuator**Class****Input Functions****Inputs**

<code>dev_handle</code>	CGIhandle for a currently open workstation
<code>val_in</code>	initial value
<code>echo_handle</code>	the <i>dev_handle</i> of the device where the echoed feedback from the input operation will be displayed

Outputs

<code>val_out</code>	output value
----------------------	--------------

Value Returned

≥ 0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Open Workstation
Sample Valuator

Description

The Request Valuator function activates the valuator device and waits until the user sets it to the requested value. When ready, the user terminates the process with a device-specific action.

See the Open Workstation parameter *work_out*[41] to determine if the device represented by *dev_handle* is capable of Valuator Input.

3.86 Reset To Defaults

Class

Control Functions

Inputs

dev_handle CGIhandle for a currently open workstation

Outputs

None

Value Returned

0 no error

-1 error occurred; actual error can be retrieved by
invoking `InquireCGIError`

Related Functions

Open Workstation

Description

The Reset To Defaults function restores the device's attribute values and operating modes to their default states. Refer to the individual Set functions for the actual defaults.

This function does not clear the view surface or internal storage such as bit-maps, raster images, or fonts, nor does it alter any state information external to the device.

Defaults are shown in the following table:

Table 3-1. Parameters Reset to Default

Parameter	Default
Select Drawing Bitmap	0 (screen)
Select Graphics Input Cursor	0 (crosshair)
Set Alpha Text Color Index	1
Set Alpha Text Font and Size	0 (font)
	0 (size)
Set Alpha Text Line Spacing	1 (single space)
Set Alpha Text Overstrike Mode	0 (off)
Set Alpha Text Pass Through Mode	0 (off)
Set Alpha Text Position	upper left corner of display surface
Set Alpha Text Quality	100
Set Alpha Text Sub/Superscript Mode	0 (off)
Set Alpha Text Underline Mode	0 (off)
Set Background Color Index	0
Set Background Mode	0 (opaque)
Set Clip Rectangle	(-32768,-32768), (32767,32767)
Set Color Table	as at open workstation
Set Cursor Text Attributes (Reverse Video)	0 (off)
Set Cursor Text Attributes (Underline)	0 (off)
Set Cursor Text Attributes (Blink)	0 (off)
Set Cursor Text Attributes (Bold)	0 (off)
Set Cursor Text Color Index	0 (background)
	1 (foreground)
Set Fill Color Index	1
Set Fill Interior Style	0 (hollow)
Set Fill Style Index	1
Set Graphics Text Alignment	0, 0 (left, base)
Set Graphics Text Character Height	minimum character height
Set Graphics Text Color Index	1
Set Graphics Text Font	1
Set Input Extent	(0,0), (32767,32767)
Set Input Mode (Choice)	1 (request)
Set Input Mode (Locator)	1 (request)
Set Input Mode (String)	1 (request)
Set Input Mode (Valuator)	1 (request)
Set Line Color Index	1
Set Line Cross Section	1
Set Line Edit Characters	CTRL-H (character delete)
	CTRL-U (line delete)
Set Line Style	1 (solid)
Set Line Width	0
Set Marker Color Index	1
Set Marker Height	minimum marker height
Set Marker Type	3 (asterisk)
Set Pen Speed	device-dependent line drawing speed
Set User Line Type	FFFFFFFF (solid)
Set Writing Mode (Plotters)	8 (overstrike)
Set Writing Mode (all other)	4 (replace)

3.87 Reverse Video Off

Class

Attribute Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
-------------------------	---------------------------------------------

Outputs

None

ValueReturned

0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

- Enter Cursor Addressing Mode
- Erase To End Of Line
- Erase To End Of Screen
- Output Cursor Addressable Text
- Reverse Video On
- Set Cursor Text Attributes
- Set Cursor Text Color Index

Description

When you are in Cursor Addressing Mode, this function displays subsequent cursor addressable text in standard video.

Note

This function is applicable only to CRT devices.

3.88 Reverse Video On**Class**

Attribute Functions

Inputs

dev_handle	CGIhandle for a currently open workstation
------------	--------------------------------------------

Outputs

None

Value Returned

0	no error
-1	error occurred; actual error can be retrieved by invoking Inquire CGIError

Related Functions

Enter Cursor Addressing Mode
 Erase To End Of Line
 Erase To End Of Screen
 Output Cursor Addressable Text
 Reverse Video Off
 Set Cursor Text Attributes
 Set Cursor Text Color Index

Description

When you are in Cursor Addressing Mode, this function displays subsequent cursor addressable text in reverse video.

Note

This function is applicable only to CRT devices.

3.89 Sample Choice

Class

Input Functions

Inputs

`dev_handle` CGI handle or a currently open workstation

Outputs

`ch_out` choice number

Value Returned

>0	no error
0	sample unsuccessful
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Open Workstation
Request Choice

Description

The Sample Choice function polls the choice device and returns a choice if one is pending.

See the Open Workstation parameter `work_out[42]` to determine if the device represented by `dev_handle` is capable of choice input.

3.90 SampleLocator**Class****Input Functions****Inputs**

dev_handle	CGI handle for a currently open workstation
xy_in[0]	initial x coordinate of locator in VDC units
xy_in[1]	initial y coordinate of locator in VDC units

Outputs

xy_out[0]	current x coordinate of locator in VDC units
xy_out[1]	current y coordinate of locator in VDC units
pressed	array element representing those buttons that have changed state from released to pressed since the last input request
released	array element representing those buttons that have changed state from pressed to released since the last input request
key_state	array element representing the current button

Value Returned

≥ 1	sample successful
0	sample unsuccessful
-1	error occurred; actual error can be retrieved by invoking Inquire CGIError

Related Functions

- Create Cursor
- Delete Cursor Description
- Display Graphics Input Cursor
- Inquire Cursor Description
- Inquire Graphics Input Cursor
- Inquire Input Extent
- Open Workstation
- Open Workstation
- Remove Graphics Input Cursor
- Select Graphics Input Cursor
- Set Input Extent

Description

The Sample Locator function returns the current position and key state of the graphics input device without waiting for operator interaction.

The two output parameters, *pressed* and *released*, represent the key transitions which have taken place between the last open workstation, sampled input, or request input function call and the current function invocation.

The parameter *key_state* represents the current state of the input device's keys. Each mouse button is represented by a single bit in the parameter *key_state*. For example, a two button mouse would have a minimum *key_state* value of 0 and a maximum of 3. In *pressed*, *released*, and *key state* the low-order bit of the word corresponds to button one, the next bit to button two, and so on.

The returned function value will be zero (0) for instances when the input device provides a non-presence status to the input driver. Devices such as light pens and graphics input tablets may return a zero value when the input stylus has been displaced from the input surface. When the returned function value is zero (0) the output parameters are defined as follows:

```
pressed = 0
released = 0
key_state = last key_state
xy_out = xy_in
```

For keyboard oriented sampled input (such as, the keyboard cursor keys) the Read Cursor Keys function is recommended.

See the Open Workstation parameter *work_out*[40] to determine if the device represented by *dev_handle* has Input Locator capabilities. This Open Workstation parameter also determines the number of triggers available for the device. (A trigger is a device-specific action caused by the user to terminate the input process.)

An application can use the Display Graphics Input Cursor and Remove Graphics Input Cursor to track a cursor on the screen.

3.91 Sample String

Class

Input Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>max_length</code>	maximum string length
<code>echo_mode</code>	echo mode: 0 don't echo input characters 1 echo input characters
<code>echo_xy[0]</code>	x coordinate of echo position in VDC units
<code>echo_xy[1]</code>	y coordinate of echo position in VDC units

Outputs

<code>string</code>	output string passed as a contiguous, null-terminated stream of bytes
---------------------	-----------------------------------------------------------------------

Value Returned

≥ 0	length of string
-1	error occurred; actual error can be retrieved by invoking <code>InquireCGIError</code>

Related Functions

Open Workstation
Request String
Set Line Edit Characters

Description

The Sample String function polls the keyboard of the requested device. Any pending input is returned until the queue is empty, a carriage return/line feed is encountered, or the input maximum string length is exceeded. The line terminators are returned. Lineedit characters function normally when using the Sample String function.

Because strings are null-terminated, the array and *max_length* must be one greater than the required number of characters to accommodate the null at the end of the string.

See the Open Workstation parameter *work_out*[43] to determine if the device represented by *dev_handle* is capable of String Input.

3.92 Sample Valuator

Class

Input Functions

Inputs

`dev_handle` CGI handle for a currently open workstation

Outputs

`val_out` current valuator value if sample successful

Value Returned

≥ 0 sample successful

-1 error occurred; actual error can be retrieved by invoking `Inquire CGIError`

Related Functions

Open Workstation
Request Valuator

Description

The Sample Valuator function returns the current value of the valuator device without waiting for operator interaction.

See the Open Workstation parameter `work_out[41]` to determine if the device represented by `dev_handle` is capable of Valuator Input.

3.93 Select Drawing Bitmap**Class**

Bitmap Functions

Inputs

<code>dev_handle</code>	CGIhandle for a currently open workstation
<code>bitmap_handle</code>	handle of bitmap to use for future operations

Outputs

None

Value Returned

≥ 0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Create Bitmap
 Inquire DrawingBitmap

Description

Use the bitmap specified by *bitmap_handle* for future graphics output primitives. All subsequent graphics primitives are drawn in the currently selected drawingbitmap.

The Select Drawing Bitmap function does not change the currently selected drawing bitmap if the input *bitmap_handle* is invalid.

3.94 Select Graphics Input Cursor

Class

Control Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>cursor_handle</code>	the <i>cursor_handle</i> (or index) to be used as the current graphics input cursor

Outputs

None

Value Returned

≥ 0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Create Cursor
Delete Cursor
Display Graphics Input Cursor
Inquire Graphics Input Cursor
Remove Graphics Input Cursor

Description

The Select Graphics Input Cursor function sets the current cursor to that specified by *cursor_handle*. The parameter *cursor_handle* may select one of six predefined cursors. The predefined cursors and their respective *cursor_handle* values are illustrated below in Figure 3-9.







CURSOR HANDLE	CURSOR NAME	SYMBOL
0	CROSSHAIR	
1	ARROW	
2	MARK	
3	POINTING HAND	
4	PALM OF HAND	
5	HOURGLASS	

Figure 3-9. *Graphics Input Cursors*

The application may also define cursors through the Create Cursor function. The *cursor_handle* returned by Create Cursor may be used with the Select Graphics Input Cursor function to specify the current graphics cursor.

If this function is called with an illegal cursor handle, no action will be taken and the currently selected graphics cursor will not be changed. There is always a currently selected graphics input cursor, even if the application has not specified one. The default cursor is the crosshair, *index0*.

Standard cursors are defined with color index 1, with an outline in color *index0*.

3.95 SetAlphaText Color Index

Class

Attribute Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>ind_in</code>	requested text color index; 0 to the device maximum

Outputs

None

Value Returned

≥ 0	selected color index
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

- Output Alpha Text
- Set Alpha Text Font And Size
- Set Alpha Text Line Spacing
- Set Alpha Text Overstrike Mode
- Set Alpha Text Quality
- Set Alpha Text Subscript/Superscript Mode
- Set Alpha Text Underline Mode
- Set Color Representation

Description

The Set Alpha Text Color Index function selects the alpha text color index for subsequent output.

At least two color indices are provided, foreground and background. Color indices range from 0 to a device-dependent maximum. See the Open Workstation parameter `work_out[13]` to determine the number of color indices available on the device represented by `dev_handle`.

Use the Set Color Representation function to change the appearance of a color index.

If the requested color index is not valid, the closest value within the range of the current device's capabilities is selected.

3.96 Set Alpha TextFontAndSize

Class

Attribute Functions

Inputs

<code>dev_handle</code>	CGIhandle for a currently open workstation
<code>font_in</code>	requested font; if the requested font is not available, font 1 (the standard font) is used; fonts 1 to 3 are fixed-spaced fonts: <ul style="list-style-type: none">1 normal/standard font (the default)2 bold; always provided for printers3 italics4 proportionally-spaced normal font5 proportionally-spaced bold6 proportionally-spaced italics>6 device-dependent
<code>size_in</code>	requested text size (1 to the device maximum); where size $n+1$ is larger (occupies more area) than size n

Outputs

<code>font_cap[0]</code>	selected text size index
<code>font_cap[1]</code>	number of horizontal character cell positions across the display surface in this font; this is ~ 1 if the selected font is a proportional font, because the character cell size is not constant
<code>font_cap[2]</code>	number of vertical character cell positions down the display surface in this font
<code>font_cap[3]</code>	number of horizontal character cell positions represented by the distance specified in <code>font_cap[6]</code>
<code>font_cap[4]</code>	number of vertical character cell positions represented by the distance specified in <code>font_cap[7]</code>

<code>font_cap[5]</code>	proportional spacing flag: 0 no 1 yes
<code>font_cap[6]</code>	width in VDC units of the number of character cells in the selected font, specified in <code>font_cap[3]</code>
<code>font_cap[7]</code>	height in VDC units of the number of character cells in the selected font, specified in <code>font_cap[5]</code>

Value Returned

≥ 0	selected font
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Inquire Alpha Text Font Capability
 Output Alpha Text
 Set Alpha Text Color Index
 Set Alpha Text Line Spacing
 Set Alpha Text Overstrike Mode
 Set Alpha Text Quality
 Set Alpha Text Subscript/Superscript Mode
 Set Alpha Text Underline Mode

Description

The Set Alpha Text And Size function sets the hardware alpha text font and size for subsequent output alpha text functions.

On printers, the resident font capability is used. Unlike graphics text, alpha text capabilities do not include font emulation.

If the parameter `font_cap[5]` is 1, the size represented by `font_cap[6]` and `font_cap[7]` may not represent the selected font. This is the case if the requested font is proportionally-spaced.

The `font_cap[6]/font_cap[3]` ratio can be used to determine the width of a character cell, including any roundoff error. This value is not accurate if the proportional spacing flag is set, because the character cell size is not constant.

The $font_cap[7]/font_cap[4]$ ratio can be used to determine the height of a character cell, including any roundoff error.

3.97 Set Alpha Text Line Spacing**Class**

Attribute Functions

Inputs

<code>dev_handle</code>	CGIhandle for a currently open workstation
<code>space_in</code>	requested line spacing; positive value in VDC units

Outputs

None

Value Returned

≥ 0	selected spacing
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

`Inquire Alpha Text Font Capability`
`Output Alpha Text`
`Set Alpha Text Color Index`
`Set Alpha Text Font And Size`
`Set Alpha Text Overstrike Mode`
`Set Alpha Text Quality`
`Set Alpha Text Subscript/Superscript Mode`
`Set Alpha Text Underline Mode`

Description

The Set Alpha Text Line Spacing function sets the vertical spacing between lines of alpha text. It determines the amount of movement down the page when it receives a line feed control character in an output alpha text string. The default is single spacing; that is, the amount of spacing between lines of alpha text is the same as the default character cell height.

Line spacing must always be a positive value. It specifies a decrement in the absolute vertical position when a line feed is encountered. You will need to update the line spacing to the character cell height of the new font (or whatever spacing is requested) whenever fonts are changed.

3.98 Set Alpha Text Overstrike Mode**Class**

Attribute Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>mode_in</code>	requested overstrike mode:
	0 off
	1 on

Outputs

None

Value Returned

≥ 0	selected mode
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Output Alpha Text
 Set Alpha Text Color Index
 Set Alpha Text Font And Size
 Set Alpha Text Line Spacing
 Set Alpha Text Quality
 Set Alpha Text Subscript/Superscript Mode
 Set Alpha Text Underline Mode

Description

The Set Alpha Text Overstrike Mode function turns overstriking on or off. The default is overstriking off. When overstriking is on, the alpha text position is not automatically advanced after each character is output; however, carriage return and line feed can still modify the current alpha text position.

The default `mode_in` is 0. If an invalid mode is requested, the default is selected.

3.99 Set Alpha Text Pass Through Mode

Class

Attribute Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>mode_in</code>	requested pass through mode:
	0 off
	1 on

Outputs

None

Value Returned

≥ 0	selected mode
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Output Alpha Text

Description

The Set Alpha Text Pass Through Mode function turns pass through mode on or off. Pass through mode enables all text to be output. Attributes such as font, color, and superscripting may not be honored. When pass through mode is in effect, text displayed does not modify the alpha text position. All characters, including control characters are sent directly to the device.

This function may be used to send device-dependent set-up strings to a particular device. The default is pass through mode off.

The default `mode_in` is 0. If an invalid mode is requested, the default is selected.

3.100 Set Alpha Text Position

Class

ControlFunctions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>x_in</code>	x coordinate of text position in VDC units
<code>y_in</code>	y coordinate of text position in VDC units

Outputs

<code>x_out</code>	x coordinate of the selected text position in VDC units
<code>y_out</code>	y coordinate of the selected text position in VDC units

Value Returned

0	no error
-1	error occurred; actual error can be retrieved by invoking <code>InquireCGIError</code>

Related Functions

`Inquire Alpha Text Position`
`Output Alpha Text`

Description

The Set Alpha Text Position function sets the alpha text position to the specified location. This specifies the position of the lower left-hand corner of the alpha text string. It is assumed that (0,0) is at the lower left-hand corner of the display surface.

The alpha position is updated only when the position is set or when the Output Alpha Text function is invoked. If the position is set at the maximum x or y extent, display of alpha text is device-dependent because characters positioned at that point would be off the display surface.

3.101 Set Alpha Text Quality

Class

Attribute Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>mode_in</code>	requested text quality, 0 to 100, where 0 is the lowest quality (draft) and 100 is the highest quality

Outputs

None

Value Returned

≥ 0	selected mode
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

- Output Alpha Text
- Set Alpha Text Color Index
- Set Alpha Text Font And Size
- Set Alpha Text Line Spacing
- Set Alpha Text Overstrike Mode
- Set Alpha Text Subscript/Superscript Mode
- Set Alpha Text Underline Mode

Description

The Set Alpha Text Quality function sets the alpha text quality to some level between draft quality and high quality. In draft quality range, small imperfections due to bidirectional printing or print head speed are acceptable. In high quality range, the output is the best possible.

The default `mode_in` is 100 (high quality). If an invalid mode is requested, the default is selected. The number of quality levels is device-dependent.

Alpha text quality is a device-dependent attribute normally associated with printers. CRTs may often have multiple alpha fonts, but do not differentiate by quality levels. On dot matrix printers, quality usually affects the number of dots used to display a character.

3.102 Set Alpha TextSub/SuperscriptMode

Class

Attribute Functions

Inputs

dev_handle	CGIhandle for a currently open workstation
mode_in	requested mode: 0 normal; subscripting and superscripting off (default) 1 subscripting on 2 superscripting on

Outputs

None

Value Returned

≥ 0	selected mode
-1	error occurred; actual error can be retrieved by invoking Inquire CGIError

Related Functions

Output Alpha Text
Set Alpha Text Color Index
Set Alpha Text Font And Size
Set Alpha Text Line Spacing
Set Alpha Text Overstrike Mode
Set Alpha Text Quality
Set Alpha Text Underline Mode

Description

The Set Alpha Subscript/Superscript Mode function sets subscripting or superscripting for subsequent alpha text. It causes output to be offset above or below the line. This is useful for footnotes, for example. The default is subscripting and superscripting off. If an invalid mode is requested, the normal mode (0) is selected.

3.103 Set Alpha Text Underline Mode

Class

Attribute Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>mode_in</code>	requested underline mode:
	0 off
	1 on

Outputs

None

Value Returned

≥ 0	selected mode
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

- Output Alpha Text
- Set Alpha Text Color Index
- Set Alpha Text Font And Size
- Set Alpha Text Line Spacing
- Set Alpha Text Overstrike Mode
- Set Alpha Text Quality
- Set Alpha Text Subscript/Superscript Mode

Description

This function turns alpha text underlining on or off. The default is underlining off. If an invalid mode is requested, the default is selected.

3.104 SetBackground ColorIndex**Class****Attribute Functions****Inputs**

dev_handle	CGIhandle for a currently open workstation
ind_in	background color index; 0 to the device maximum

Outputs

None

Value Returned

≥ 0	selected color index
-1	error occurred; actual error can be retrieved by invoking Inquire CGIError

Related Functions

Clear Workstation
 Inquire Color Representation
 Inquire Drawing Bitmap
 Open Workstation
 Select Drawing Bitmap
 Set Color Representation

Description

The Set Background Color Index function sets the background color of the device to the requested index. On some devices, this change may not appear until the next time the Clear Workstation function is invoked. If the index is not valid, no change will be made in the background index. In all cases, the selected color index is returned.

See the Open Workstation parameter *work_out*[13] to determine the number of color indices available on the device represented by *dev_handle*.

Use the Set Color Representation function to change the appearance of a color index.

Note

This function is not applicable on plotters.

3.105 SetBackgroundMode**Class**

Control Functions

Inputs

<code>dev_handle</code>	CGIhandle for a currently open workstation
<code>mode</code>	requested background mode:
	0 opaque
	1 transparent

Outputs

None

Value Returned

≥ 0	selected mode
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Clear Workstation
 Copy Bitmap
 Inquire Background Mode
 Output Arc
 Output Bar
 Output Byte Pixel Array
 Output Circle
 Output Ellipse
 Output Elliptical Arc
 Output Elliptical Pie Slice
 Output Filled Area
 Output Graphics Text
 Output Integer Pixel Array
 Output Pie Slice
 Output Polyline
 Output Polymarker
 Set Writing Mode

Description

The Set Background Mode function determines how background pixels affect the appearance of all output primitives. Any time a pixel in the source output primitive (line, text, bitmap, etc.) is equal to the current background color and the background mode is transparent, the corresponding pixel in the output bitmap is untouched.

If *mode* is 0 (opaque), the current background index is written into the background pixels using the current writing mode. If *mode* is 1 (transparent), background pixels are affected only by Clear Workstation.

The default mode at Open Workstation is 0 (opaque).

An invalid *mode* causes no change of mode.

3.106 Set Clip Rectangle

Class**Attribute Functions****Inputs**

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>clip_rectangle[0]</code>	<i>x</i> coordinate, in VDC units, of the first end point of a diagonal that describes the clipping rectangle
<code>clip_rectangle[1]</code>	<i>y</i> coordinate, in VDC units, of the first end point of a diagonal that describes the clipping rectangle
<code>clip_rectangle[2]</code>	<i>x</i> coordinate, in VDC units, of the second end point of a diagonal that describes the clipping rectangle
<code>clip_rectangle[3]</code>	<i>y</i> coordinate, in VDC units, of the second end point of a diagonal that describes the clipping rectangle

Outputs

None

Value Returned

≥ 0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

- Create Bitmap
- Inquire Clip Rectangle
- Output Arc
- Output Bar
- Output Byte Pixel Array
- Output Cell Array
- Output Circle
- Output Ellipse
- Output Elliptical Arc
- Output Elliptical Pie Slice
- Output Filled Area
- Output Graphics Text
- Output Integer Pixel Array
- Output Pie Slice
- Output Polyline
- Output Polymarker

Description

The Set Clip Rectangle function changes the clipping region of the CGI output to a region specified by the input parameter *clip_rectangle*. The actual clipping limits are the intersection of the clipping rectangle and the limits of the currently selected bitmap.

Portions of graphics primitives outside of the clip rectangle are not displayed. The clip rectangle applies to the currently selected bitmap of the specified open workstation.

The default clip rectangle at open workstation is -32768 to 32767 in both *x* and *y*.

3.107 Set Color Representation**Class****Attribute Functions****Inputs**

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>ind_in</code>	requested color index
<code>rgb[0]</code>	red color intensity (in tenths of percent, 0 to 1000)
<code>rgb[1]</code>	green color intensity (in tenths of percent, 0 to 1000)
<code>rgb[2]</code>	blue color intensity (in tenths of percent, 0 to 1000)

Outputs

<code>rgb_out[0]</code>	selected red color intensity (in tenths of percent, 0 to 1000)
<code>rgb_out[1]</code>	selected green color intensity (in tenths of percent, 0 to 1000)
<code>rgb_out[2]</code>	selected blue color intensity (in tenths of percent, 0 to 1000)

Value Returned

≥ 0	selected color index
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

`Inquire Color Representation`
`Open Workstation`
`Set Color Table`

Description

The Set Color Representation function is used to map an index to a new color.

At least two color indices are provided: foreground and background. If the requested index is outside of the device's capabilities, the closest device index is set. If a color intensity of less than 0 is requested, it is mapped to 0. If a color intensity greater than 1000 is requested, it is mapped to 1000.

To change the appearance of a color index, you must select the desired levels of the three color components (red, green and blue) that make up the index using this function. This method can be used to create non-default colors such as brown or orange. The new color will be visible only on devices that support color definition.

See the Open Workstation parameters *work_out*[13] and *work_out*[39] to determine the number of color indices and colors available on the device represented by *dev_handle*.

3.108 Set Color Table

Class

Attribute Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>s_index</code>	starting index of color table to start loading
<code>n</code>	number of color table entries to be loaded
<code>rgb[n][3]</code>	<i>rgb</i> color intensities to be loaded (in tenths of percent 0-1000)

Outputs

None

Value Returned

≥ 0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

`Inquire Color Representation`
`Open Workstation`
`Set Color Representation`

Description

The color intensity elements specified by the *n* by 3 array *rgb* are loaded, in the order specified, into consecutive locations of the color table beginning at the starting index, *s_index*. Only the specified color table entries are changed. The effect of changes in the color table on any existing graphics that use the affected indices is device dependent.

Out of range indices cause no change.

See the `Open Workstation` parameters `work_out[13]` and `work_out[39]` to determine the number of color indices and colors available on the device represented by *dev_handle*.

3.109 Set CursorText Attributes

Class

Attribute Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>req_att[0]</code>	requested reverse video mode: 0 disable reverse video 1 enable reverse video 2 do not change current state 3 toggle reverse video status
<code>req_att[1]</code>	requested underline cursor text mode: 0 disable underline cursor text 1 enable underline cursor text 2 do not change current state 3 toggle underline cursor text status
<code>req_att[2]</code>	requested blink text mode: 0 disable blink cursor text 1 enable blink cursor text 2 do not change current state 3 toggle blink cursor text status

<code>req_att[3]</code>	requested bold cursor text mode:
	0 disabled bold cursor text
	1 enable bold cursor text
	2 do not change current state
	3 toggle bold cursor text status

Outputs

<code>sel_att[0]</code>	selected reverse video mode:
	1 enabled
	0 disabled

<code>sel_att[1]</code>	selected underline cursor text mode:
	1 enabled
	0 disabled

<code>sel_att[2]</code>	selected blink text mode:
	1 enabled
	0 disabled

<code>sel_att[3]</code>	selected bold cursor text mode:
	1 enabled
	0 disabled

Value Returned

0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

- Enter Cursor Addressing Mode
- Erase To End Of Line
- Erase To End Of Screen
- Output Cursor Addressable Text
- Reverse Video Off
- Reverse Video On
- Set Cursor Text Color Index

Description

The Set Cursor Text Attributes function allows the attributes of reverse video, underline, blink, and bold to be set for subsequent cursor addressable text. The reverse video mode can be set in this function or in the Reverse Video On and Reverse Video Off functions. Whichever mode was invoked last will be used as the reverse video attribute for subsequent cursor addressable text.

This function can also be used to inquire the current status by setting all modes to 2 (do not change current state). The current state is then returned in *sel_att*.

3.110 Set CursorTextColorIndex**Class****Attribute Functions****Inputs**

<code>dev_handle</code>	CGIhandle for a currently open workstation
<code>fore_requested</code>	color index of the foreground of subsequent output cursor text; the default is 1
<code>back_requested</code>	color index of the background of subsequent output cursor text; the default is 0

Outputs

<code>fore_selected</code>	selected color index for cursor text foreground
<code>back_selected</code>	selected color index for cursor text background

Value Returned

0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Enter Cursor Addressing Mode
 Erase To End Of Line
 Erase To End Of Screen
 Output Cursor Addressable Text
 Reverse Video Off
 Reverse Video On
 Set Cursor Text Attributes

Description

The Set Cursor Text Color Index function sets the foreground and background colors for cursor addressable text.

If an invalid color index is specified, it is mapped to the index closest to the current device's capabilities.

3.111 Set Deferral Mode

Class

Control Functions

Inputs

dev_handle	CGI handle for currently open workstation
mode	deferral mode for CGI function calls: 0 execute at some time in the future (ASTI) (default) 1 defer execution until the next user interaction (BNI) 2 execute as soon as possible (ASAP)

Outputs

None

Value Returned

0	no error
-1	error occurred; actual error can be retrieved by invoking Inquire CGIError

Related Functions

Inquire Deferral Mode
Reset To Defaults

Description

The Set Deferral Mode function determines when CGI function calls are executed. When deferral mode is ASTI, the device driver can postpone execution until it is convenient (that is, it can buffer output).

When deferral mode is ASAP, each CGI function is executed as it is processed.

When deferral mode is BNI, the CGI functions that do not have output parameters are buffered by the CGI and will be executed in chronological order when a function requiring output is encountered.

Functions that are deferred for later execution always return a value of zero (0). This return value has no relevance to normal returns, and is, therefore, meaningless. However, the application can check the error return value from the CGI by invoking Inquire CGI Error which will have the same effect as a user interaction routine (that is, it will bring all devices up to date).

The following functions will be deferred when BNI is selected:

Application Data	Select Graphics Input Cursor
Clear Workstation	Set Alpha Text Color Index
Copy Bitmap	Set Alpha Text Line Spacing
Cursor Down	Set Alpha Text Overstrike Mode
Cursor Home	Set Alpha Text Pass Through Mode
Cursor Left	Set Alpha Text Quality
Cursor Right	Set Alpha Text Sub/Superscript Mode
Cursor Up	Set Alpha Text Underline Mode
Delete Bitmap	Set Background Color Index
Delete Cursor	Set Background Mode
Direct Cursor Address	Set Clip Rectangle
Display Graphics Input Cursor	Set Color Table
Enter Cursor Addressing Mode	Set Fill Area Representation
Erase To End Of Line	Set Fill Color
Erase To End Of Screen	Set Fill Interior Style
Exit Cursor Addressing Mode	Set Fill Style Index
Hard Copy	Set Graphics Text Color Index
Message	Set Graphics Text Font
Output Arc	Set Graphics Text Representation
Output Bar	Set Graphics Text String Baseline Rotation
Output Cell Array	Set Input Extent
Output Circle	Set Line Edit Characters
Output Cursor Addressable Text	Set Pen Speed
Output Ellipse	Set Line Color Index
Output Elliptical Arc	Set Line Cross Section
Output Elliptical Pie Slice	Set Line Representation
Output Filled Area	Set Line Type
Output Graphics Text	Set Line Width
Output Pie Slice	Set Marker Color Index
Output Polyline	Set Marker Height
Output Polymarker	Set Marker Representation
Remove Graphics Input Cursor	Set Marker Type
Reset To Defaults	Set User Line Type
Reverse Video Off	Set Writing Mode
Reverse Video On	Update Workstation
Select Drawing Bitmap	

3.112 SetFillArea Representation

Class

Attribute Functions

Inputs

- dev_handle** CGI handle for a currently open workstation
- n** the number of elements in the array *attrib*
- attrib[0]** fill area interior style:
 - 0 hollow
 - 1 solid
 - 2 pattern
 - 3 hatch
 - 4 bitmap
- attrib[1]** fill area color index
- attrib[2]** fill area style; meaning depends on fill area interior style (*attrib[0]*):

attrib[0] value	attrib[2] value
hollow	meaningless
solid	meaningless
pattern	pattern index
hatch	hatch index
bitmap	bitmap handle; full bitmap used as fill pattern

Outputs

None

Value Returned

≥ 0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGI Error</code>

Related Functions

`Inquire Fill Area Representation`
`Open Workstation`
`Output Bar`
`Output Circle`
`Output Ellipse`
`Output Elliptical Pie Slice`
`Output Filled Area`
`Output Pie Slice`
`Set Fill Color Index`
`Set Fill Interior Style`
`Set Fill Style Index`

Description

The `Set Fill Area Representation` function uses the specified attributes in future area primitives.

When only part of the input attribute array is passed in rather than the whole array, the effect will be as if one had invoked an inquiry of the full array, copied onto that inquiry the part of the array actually passed in, and then invoked the `Set` function with the entire array.

Only exactly correct attribute settings are accepted via the attribute array. If a requested attribute setting is not exactly available, the current setting of the attribute is not changed. This behavior differs from the individual attribute setting functions, where defaults and adjustments are available. The intent is that the setting functions be used to set all the attributes associated with a given primitive to values derived from an earlier inquiry.

See the `Open Workstation` parameters `work_out[11]`, `work_out[12]`, and `work_out[13]` for device-specific information regarding this function.

3.113 SetFillColorIndex

Class

Attribute Functions

Inputs

dev_handle	CGI handle for a currently open workstation
ind_in	requested fill color index (0 to device maximum)

Outputs

None

Value Returned

≥ 0	selected color index
-1	error occurred; actual error can be retrieved by invoking <code>InquireCGIError</code>

Related Functions

Inquire Color Representation
Inquire Fill Area Representation
Open Workstation
Output Bar
Output Circle
Output Ellipse
Output Elliptical Pie Slice
Output Filled Area
Output Pie Slice
Set Background Color Index
Set Color Representation
Set Fill Area Representation
Set Fill Interior Style
Set Fill Style Index

Description

The Set Fill Color Index function determines the color to be used for filling bars, pie slices, ellipses, elliptical pie slices, filled areas, and circles.

At least two color indices are provided: foreground and background. Color indices range from 0 to a device-dependent maximum.

Use the Set Color Representation function to change the appearance of a color index.

If the color specified is invalid, the closest value in the range is chosen. In all cases, the selected color index is returned.

See the Open Workstation parameter *work_out*[13] to determine the number of color indices available to the device represented by *dev_handle*.

3.114 SetFillInteriorStyle

Class

Attribute Functions

Inputs

<code>dev_handle</code>	CGIhandle for a currently open workstation
<code>styleIn</code>	requested fill interior style: 0 hollow 1 solid 2 pattern 3 hatch 4 bitmap

Outputs

None

Value Returned

≥ 0	selected style
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

`Inquire Fill Area Representation`
`Output Bar`
`Output Circle`
`Output Ellipse`
`Output Elliptical Pie Slice`
`Output Filled Area`
`Output Pie Slice`
`Set Fill Area Representation`
`Set Fill Color Index`
`Set Fill Style Index`

Description

The Set Fill Interior Style function sets the style of fill to be used for filled areas, bars, pie slices, ellipses, elliptical pie slices, and circles.

When you select hollow, the area is outlined with a solid line in the current fill color. When you select solid, the area is filled in the current color. When you select pattern, hatch, or bitmap this function may be used to select a particular fill pattern.

The default style is hollow. If the requested style is invalid, the default is used.

Solid, hatch, and pattern and bitmap filled areas are not outlined.

3.115 SetFillStyle Index

Class

Attribute Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>ind_in</code>	requested fill style index for pattern or hatch fill

Outputs

None

Value Returned

≥ 0	selected index
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

- Inquire Fill Area Representation
- Open Workstation
- Output Bar
- Output Circle
- Output Ellipse
- Output Elliptical Pie Slice
- Output Filled Area
- Output Pie Slice
- Set Fill Area Representation
- Set Fill Color Index
- Set Fill Interior Style

Description

The Set Fill Style Index function selects a fill style based on the fill interior style. This index has no effect if the interior style is either hollow or solid. If the requested index is not available, index 1 is used.

The index references a hatch style if the fill interior style is Hatch, or a pattern if the fill interior style is Pattern.

The six pre-defined hatch styles are listed below and illustrated in Figure 3-10.

- | | |
|----|-----------------------------------|
| 1 | narrow spaced +45° lines |
| 2 | medium spaced +45° lines |
| 3 | widely spaced +45° lines |
| 4 | narrow spaced +45° and -45° lines |
| 5 | medium spaced +45° and -45° lines |
| 6 | widely spaced +45° and -45° lines |
| >6 | device-dependent |

The thirty-three pre-defined pattern styles are illustrated in Figure 3-11.

There is no difference between hatch and pattern styles on some devices. For example, asking for a hatch style of 3 may result in the same output as asking for a pattern style of 3.

See the Open Workstation parameters *work_out*[11] and *work_out*[12] to determine the number of patterns and hatch styles available to the device represented by *dev_handle*.

If the fill interior style is bitmap, *ind_in* is a bitmap handle.

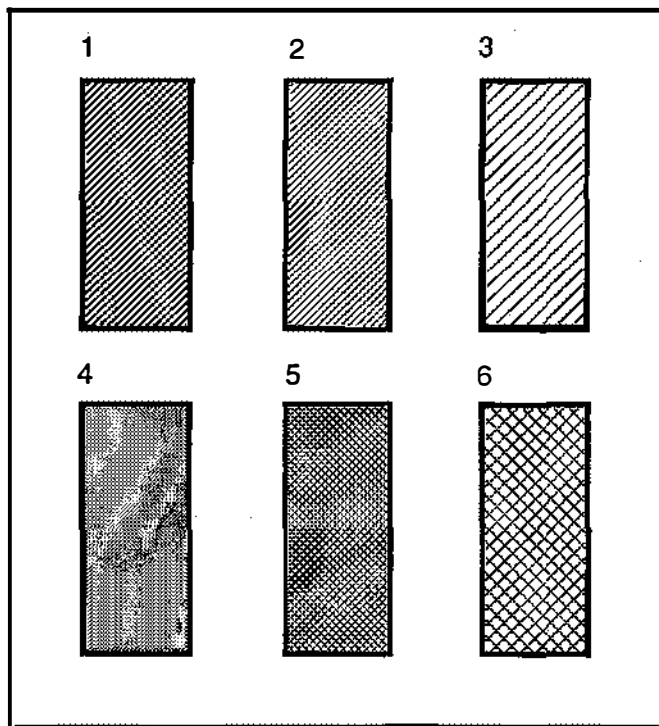


Figure 3-10. *Hatch Styles*

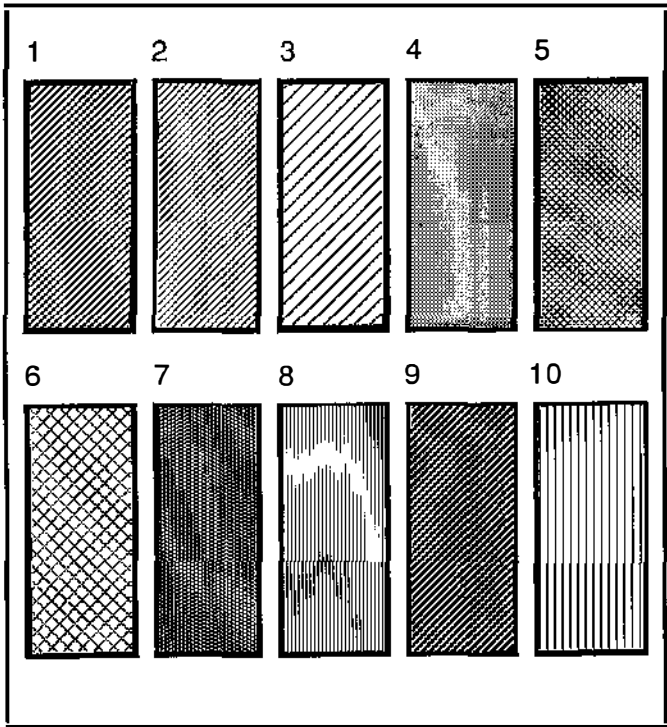


Figure 3-11. *Pattern Styles*

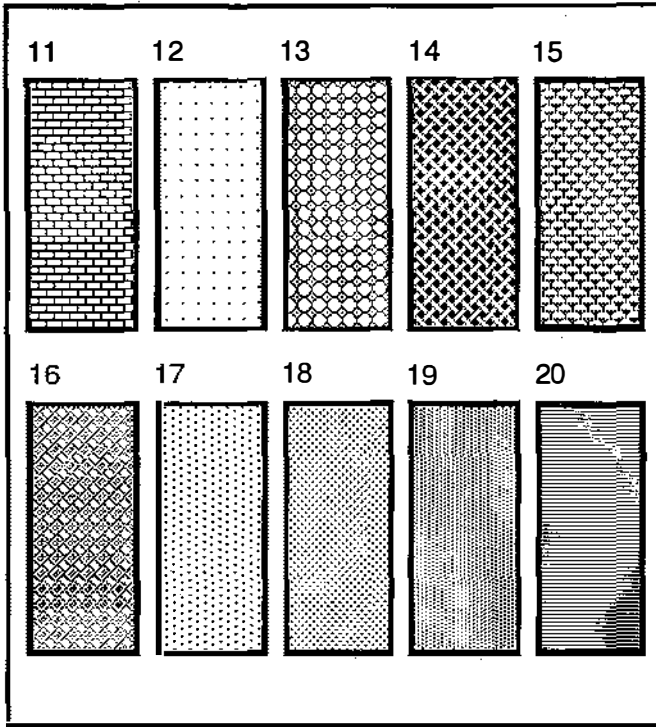


Figure 3-11. Pattern Styles (continued)

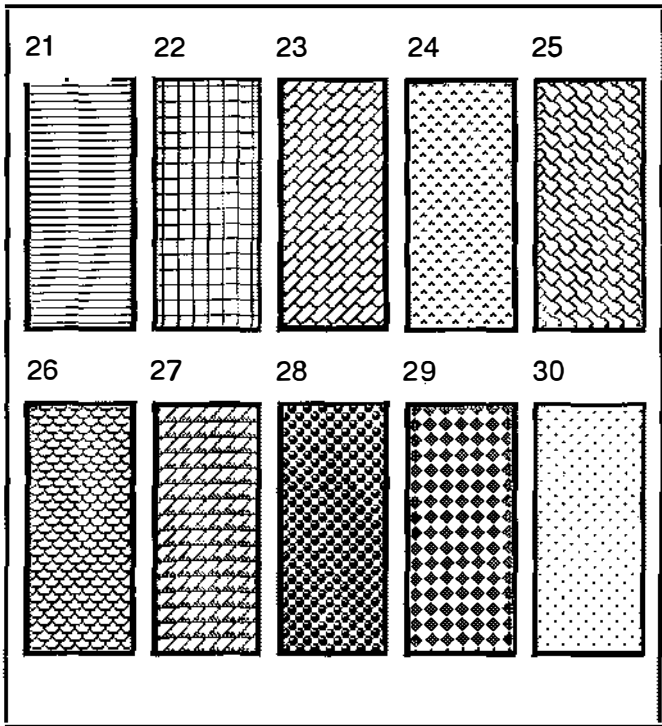


Figure 3-11. *Pattern Styles* (continued)

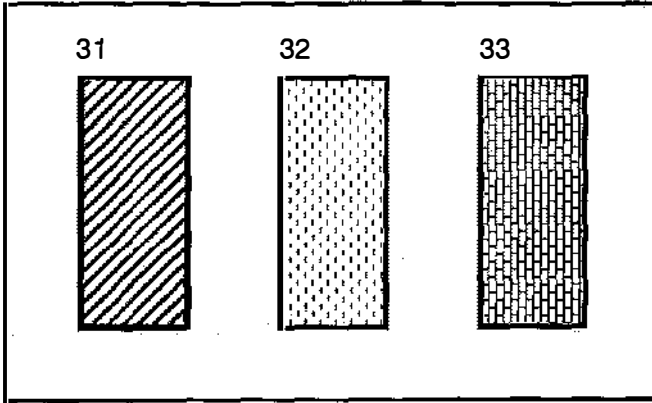


Figure 3-11. *Pattern Styles* (continued)

3.116 SetGraphicsTextAlignment**Class****Attribute Functions****Inputs**

<code>dev_handle</code>	CGIhandle for a currently open workstation
<code>hor_in</code>	requested horizontal alignment: 0 left 1 center 2 right
<code>vert_in</code>	requested vertical alignment: 0 base 1 half 2 cap 3 bottom 4 top

Outputs

<code>hor_out</code>	selected horizontal alignment
<code>vert_out</code>	selected vertical alignment

Value Returned

0	no error
-1	error occurred; actual error can be retrieved by invoking <code>InquireCGIError</code>

Related Functions

Inquire Graphics Text Extent
Inquire Graphics Text Representation
Open Workstation
Output Graphics Text
Set Graphics Text Character Height
Set Graphics Text Color Index
Set Graphics Text Font
Set Graphics Text Representation
Set Graphics Text String Baseline Rotation

Description

The current text alignment is set to the requested input parameters. The text alignment specifies the placement of the character string relative to the x, y alignment point. See the Output Graphics Text function for more information.

Text is aligned with respect to a text extent rectangle. The text extent rectangle is derived by joining the character cells of the characters in the string. The Graphics Text Alignment attribute controls the positioning of the text extent rectangle in relation to the text position (see Figures 3-12 and 3-13).

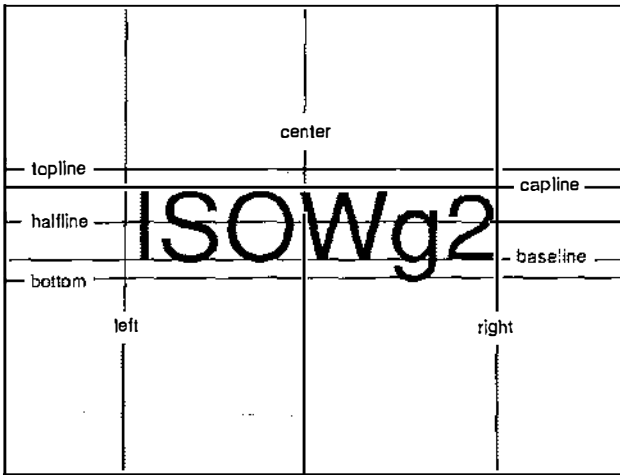


Figure 3-12. *Text Alignment Positions, #1*

†left top	center† top	right top†
†left cap	center† cap	right cap†
†left half	center† half	right half†
†left base	center† base	right base†
†left bottom	center† bottom	right bottom†

Figure 3-13. *Text Alignment Positions, #2*

The horizontal component of text alignment has three possible values: *left*, *center*, and *right*. If the horizontal component is *left*, the left side of the text extent rectangle passes through the text position. If the value is *right*, the right side of the text extent rectangle passes through the text position. If the value is *center*, the text position lies midway between the left and right sides of the text extent rectangle. The vertical component of text alignment has five possible values: *top*, *cap*, *half*, *base*, and *bottom*. A vertical alignment value of *top*, *cap*, *half*, *base*, or *bottom* causes the text to be moved such that the corresponding defining line of the text extent rectangle passes through the text position.

See the Open Workstation parameter *work_out*[48] to determine if the device represented by *dev_handle* is capable of graphics text alignment.

3.117 SetGraphics TextCharacterHeight

Class

Attribute Functions

Inputs

dev_handle	CGI handle for a currently open workstation
rq_height	requested character height in VDC units

Outputs

char_width	selected character width in VDC units
cell_width	selected character cell width in VDC units
cell_height	selected character cell height in VDC units

Value Returned

≥ 0	selected character height in VDC units
-1	error occurred; actual error can be retrieved by invoking <code>InquireCGIError</code>

Related Functions

- Inquire Graphics Text Extent
- Inquire Graphics Text Representation
- Open Workstation
- Output Graphics Text
- Set Graphics Text Alignment
- Set Graphics Text Color Index
- Set Graphics Text Font
- Set Graphics Text Representation
- Set Graphics Text String Baseline Rotation

Description

The Set Graphics Text Character Height function sets the size of subsequent graphics text.

The specified height is that of the actual character (baseline to top of tallest character), not the character cell. If the requested size is outside of device capabilities, the closest available size on the device is used. If the requested character height does not map exactly to a device size, the largest character height that does not exceed the requested size is used.

When changing a set of text attributes that includes Set Graphics Text Height, the text height should be the last attribute set in order to ensure the most accurate representation. This is particularly important when using software fonts.

The default size permits at least 80 characters to be displayed horizontally across the display surface and 24 characters to be displayed vertically down the display surface. The font coordinate system is illustrated in Figure 3-14 below. The character body encloses all of the drawn parts of all characters in the font. That is, no descender goes below the bottom line and no accent mark or oversized character goes above the top line.

The left and right edges of the character body may be defined on a per-character basis to accommodate variable widths, and proportional spacing. The character cell exceeds the actual graphics character width and height as necessary to provide adequate white space between characters, such that text is readable and adequately separated when adjacent character cells are flush in both the horizontal and vertical directions.

The Text Height specifies the VDC distance between the cap line and base line of the font (see Figure 3-14 below).

See the Open Workstation parameter *work_out*[5] to determine the number of graphics text character heights available on the device represented by *dev_handle*. See the Open Workstation parameters *work_out*[60] and *work_out*[61] to determine the minimum and maximum graphics text character heights on the device represented by *dev_handle*.

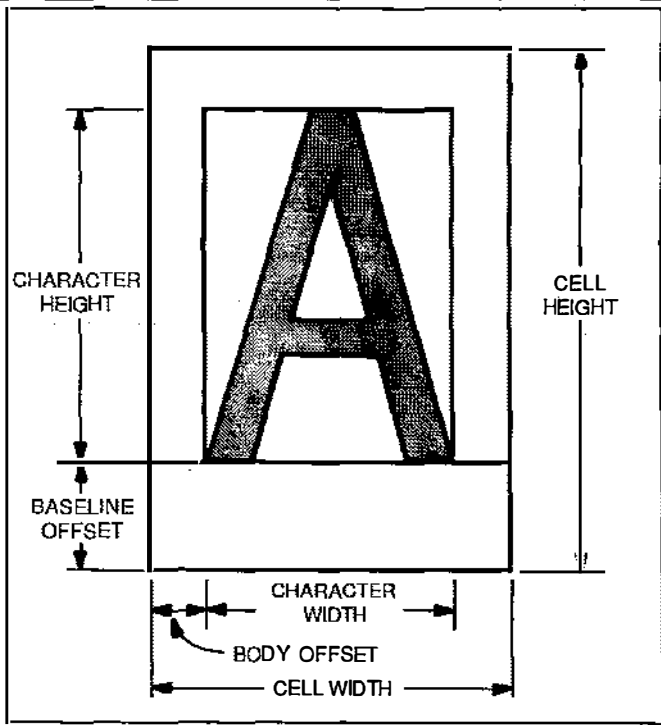


Figure 3-14. *The Font Coordinate System*

3.118 Set Graphics Text Color Index**Class**

Attribute Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>ind_in</code>	requested text color index; 0 to the device maximum

Outputs

None

Value Returned

≥ 0	selected color index
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Inquire Color Representation
 Inquire Graphics Text Representation
 Open Workstation
 Output Graphics Text
 Set Color Representation
 Set Graphics Text Alignment
 Set Graphics Text Character Height
 Set Graphics Text Font
 Set Graphics Text Representation
 Set Graphics Text String Baseline Rotation

Description

The Set Graphics Text Color Index function sets the graphics text color index.

At least two color indices are provided: foreground and background. Color indices range from 0 to a device-dependent maximum.

Use the Set Color Representation function to change the appearance of a color index.

SCO CGI Programmer's Guide

If the color index requested is not valid, the closest value within the range of the current device's capabilities is selected. In any case, the selected color index is returned.

See the Open Workstation parameter *work_out*[13] to determine the number of color indices available on the device represented by *dev_handle*.

3.119 Set Graphics Text Font**Class**

Attribute Functions

Inputs

<code>dev_handle</code>	CGIhandle for a currently open workstation
<code>font_in</code>	requested font index

Outputs

None

Value Returned

≥ 0	selected font
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Inquire Graphics Text Extent
 Inquire Graphics Text Font Description
 Inquire Graphics Text Representation
 Open Workstation
 Output Graphics Text
 Set Graphics Text Alignment
 Set Graphics Text Character Height
 Set Graphics Text Color Index
 Set Graphics Text Representation
 Set Graphics Text String Baseline Rotation

Description

The text font index is used to select a particular font for subsequent Graphics Text outputs.

Every output device supports at least one font that is able to generate a graphical representation of the ASCII character set. This is font number 1. Other hardware fonts are indexed 2, 3, ... n (where n is the number of hardware fonts available). If the font driver is installed and software fonts are available, their indices start at $n+1$.

Notice that the index of a particular software font depends on the configuration of the system being used.

After changing the text font, verify that you have the expected text size.

See the Open Workstation parameter *work_out*[10] to determine the number of graphics text fonts available on the device represented by *dev_handle*. You can then use the Inquire Graphics Text Font Description function (in conjunction with the Set Graphics Text Font function) to determine the characteristics of the font represented by each index.

3.120 Set Graphics Text Representation

Class

Attribute Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>n</code>	the number of elements in the array <i>attrib</i>
<code>attrib[0]</code>	requested graphics text font
<code>attrib[1]</code>	requested graphics text color
<code>attrib[2]</code>	requested graphics text baseline angle; in tenths of degrees, 0 to 3599
<code>attrib[3]</code>	requested horizontal alignment:
	0 left
	1 center
	2 right
<code>attrib[4]</code>	requested vertical alignment:
	0 base
	1 half
	2 cap
	3 bottom
	4 top
<code>attrib[5]</code>	ignored
<code>attrib[6]</code>	requested character height in VDC units
<code>attrib[7]</code>	ignored

attrib[8] ignored

Outputs

None

Value Returned

≥0 no error

-1 error occurred; actual error can be retrieved by
invoking `Inquire CGIError`

Related Functions

- Inquire Graphics Text Extent
- Inquire Graphics Text Representation
- Open Workstation
- Output Graphics Text
- Set Graphics Text Alignment
- Set Graphics Text Character Height
- Set Graphics Text Color Index
- Set Graphics Text Font
- Set Graphics Text String Baseline Rotation

Description

The `Set Graphics Text Representation` function may be used to set the current graphics text representation in a single call.

When only part of the input attribute array is passed in rather than the whole array, the effect will be as if one had done an inquiry of the full array, copied onto that inquiry the part of the array actually passed in, and then invoked the `Set` function with the entire array.

Only exactly correct attribute settings are accepted via the attribute array. If a requested attribute setting is not exactly available, the current setting of the attribute is not changed. This behavior differs from the individual attribute setting functions, where defaults and adjustments are available. The intent is that the setting functions be used to set all the attributes associated with a given primitive to values derived from an earlier inquiry.

See the `Open Workstation` parameters `work_out[5]`, `work_out[10]`, `work_out[13]`, `work_out[36]`, `work_out[48]` for information regarding this function.

3.121 Set Graphics TextString Baseline Rotation**Class****Attribute Functions****Inputs**

<code>dev_handle</code>	CGIhandle for a currently open workstation
<code>ang_in</code>	requested angle of rotation of character baseline; in tenths of degrees, 0 to 3599

Outputs

None

Value Returned

≥ 0	selected angle
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Inquire Graphics Text Extent
Inquire Graphics Text Representation
Open Workstation
Output Graphics Text
Set Graphics Text Alignment
Set Graphics Text Character Height
Set Graphics Text Color Index
Set Graphics Text Font
Set Graphics Text Representation

Description

The `Set Graphics Text String Baseline Rotation` function sets the baseline rotation of graphics text. The entire string of text is rotated (rather than each character separately) specified by the angle of rotation.

The angle specification assumes that 0° is at 3 o'clock, with angles increasing in the counterclockwise direction. If the requested angle is outside of the range (0-3599), a character baseline of 0° is used.

Fonts will not look the same when they are rotated as they do when they are not rotated if the current output device does not have square pixels.

See the Open Workstation parameters *work_out*[3] and *work_out*[4] to determine the aspect ratio of the device's pixels.

3.122 SetInput Extent**Class**

Input Functions

Inputs

dev.handle	CGIhandle for a currently open workstation
rectangle[0]	x coordinate of the first corner of the input device's extent in VDC units
rectangle[1]	y coordinate of the first corner of the input device's extent in VDC units
rectangle[2]	x coordinate of the second corner of the input device's extent in VDC units
rectangle[3]	y coordinate of the second corner of the input device's extent in VDC units

Outputs

None

Value Returned

≥ 0	no error
-1	error occurred; actual error can be retrieved by invoking Inquire CGIError

Related Functions

Inquire Input Extent
Request Locator
Sample Locator

Description

The Set Input Extent function sets the extents of the input device so that the coordinates returned from a locator function correspond to the extents of the output device.

The two corners define the rectangular extents within which the input device will return all input coordinates. The default extent is (0,0) and (32767, 32767).

To set the input extent to a canonical rectangle that corresponds to the output display surface, first set a clip rectangle on the output device the same size as the input extent rectangle, then inquire on the output device's clip rectangle. The returned rectangle will be canonical. Next, set the input extent rectangle to the result of the clip rectangle inquiry, then reset your clip rectangle. You now have a canonical input extent rectangle.

As an example of how this might be done, the following pseudocode segment opens up a display and a mouse and sets the mouse's input extents to the canonical rectangle that is the lower left quarter of the display surface. The output device's clip rectangle is restored to full display size after the setting of the input extent.

```
.  
.
OPEN_WORKSTATION (display_in, display_handle, display_workout);
OPEN_WORKSTATION (mouse_in, mouse_handle, mouse_workout);
rectangle[0]=0;
rectangle[1]=0;
rectangle[2]= display_workout[51]/2;
rectangle[3]= display_workout[52]/2;
SET_CLIP_RECTANGLE (display_handle, rectangle);
INQUIRE_CLIP_RECTANGLE (display_handle, rectangle);
SET_INPUT_EXTENT (mouse_handle, rectangle);

/* Restore the output device's clip rectangle to full screen */
rectangle[2]= display_workout[51];
rectangle[3]= display_workout[52];
SET_CLIP_RECTANGLE (display_handle, rectangle);
.  
.
```

3.123 Set Line Color Index**Class**

Attribute Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>ind_in</code>	requested color index; 0 to the device maximum

Outputs

None

Value Returned

≥ 0	selected color index
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Inquire Color Representation
 Inquire Line Representation
 Open Workstation
 Output Arc
 Output Elliptical Arc
 Output Polyline
 Set Color Representation
 Set Line Cross Section
 Set Line Type
 Set Line Representation
 Set User Line Type

Description

The Set Line Color Index function sets the color index for subsequent polylines and arcs.

At least two color indices are provided: foreground and background. Color indices range from 0 to a device-dependent maximum.

Use the Set Color Representation function to change the appearance of a color index.

If the color specified is invalid, the closest value in range is chosen. In any case, the selected color index is returned.

See the Open Workstation parameter *work_out*[13] to determine the number of color indices available to the device represented by *dev_handle*.

3.124 SetLine Cross Section**Class**

Attribute Functions

Inputs

dev_handle	CGIhandle for a currently open workstation
bitmap_handle	handle of the bitmap to use when drawing line primitives

Outputs

None

Value Returned

≥0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Inquire Line Representation
 Output Polyline
 Set Background Mode
 SetLine Color Index
 SetLine Representation
 SetLine Type
 SetLine Width
 Set User Line Type

Description

Lines have a visually square cross section with a width specified by the Set Line Width function. The input argument, *bitmap_handle*, specifies whether a solid pattern, a pre-defined pattern, or a user-defined bitmap pattern will be used to fill fat polylines. The range of values and meanings for the argument *bitmap_handle* is:

- 0 fill fat lines with the solid current line color;
- 1-*n* fill fat lines with the corresponding pattern index, in the current line color (where *n* is the number of pre-defined patterns supported by the device);
- >*n* bitmap handle of bitmap used to fill fat lines.

Fat dashed lines have the dash pattern scaled by the line width. Line cross section is ignored for lines of width zero. If the current background mode is opaque, the background part of dashed lines is drawn with the Solid Fill background color.

If an invalid *bitmap_handle* is specified, no change is made and the Invalid Range Error is returned.

3.125 SetLine Edit Characters

Class

Attribute Functions

Inputs

dev_handle	CGIhandle for a currently open workstation
line_del	character to use to delete previous line; CTRL-U (NAK) is used as the default
char_del	character to use to delete previous character; CTRL-H (Backspace) is used as the default

Outputs

None

Value Returned

0	no error
-1	error occurred; actual error can be retrieved by invoking Inquire CGIError

Related FunctionsRequest String
Sample String**Description**

The Set Line Edit Characters function sets the current line editing characters. They apply to the Request String and Sample String functions only.

3.126 Set Line Representation

Class

Attribute Functions

Inputs

dev_handle	CGIhandle for a currently open workstation
n	the number of elements in the array <i>attrib</i>
attrib[0]	linetype: <ul style="list-style-type: none"> 0 user-specified; see <i>attrib</i>[2], <i>attrib</i>[3] 1 solid 2 longdashed 3 dotted 4 dashed-dotted 5 medium dashed 6 dashed with 2 dots 7 short dashed >7 device-dependent
attrib[1]	linecolorindex
attrib[2], attrib[3]	specifies a 32-bit user-defined line type; <i>attrib</i> [2] is the low-order half, <i>attrib</i> [3] is the high-order half

attrib[4]	line width in VDC units
attrib[5]	line cross section:
0	solid
1- <i>n</i>	fill style index, when fill style interior is pattern or hatch
> <i>n</i>	bitmap handle of user-defined cross section

Outputs

None

Value Returned

≥0	no error
-1	error occurred; actual error can be retrieved by invoking Inquire CGIError

Related Functions

Inquire Line Representation
 Open Workstation
 Output Arc
 Output Elliptical Arc
 Output Polyline
 Set Line Color Index
 Set Line Cross Section
 Set Line Type
 Set Line Width
 Set User Line Type

Description

The Set Line Representation function uses the specified attributes in future line primitives. This function may be used instead of the following individual attribute setting functions:

Set Line Type
 Set Line Color
 Set User Line Type
 Set Line Width

The values of the input parameters *attrib*[2] and *attrib*[3] are valid only if *attrib*[0] is 0. In the C language, addressing of these two 16-bit elements may be accomplished through type casting; for example:

```
((long*)(&attrib[2]))
```

It is the responsibility of the individual language bindings to do whatever word or byte reordering is necessary to accomplish the above assignment.

The line color attribute sets the line color for subsequent line primitives.

The line type attribute behaves as specified in Set Line Type. If the specified line type is 0 (user-specified), *attrib*[2], *attrib*[3] specifies the dash pattern to be used for polylines. An example of how the application may set the dash pattern in C Language follows:

```
dash_patt = 0xF0F0A0A0L;  
*((long*)(&attrib[2])) = dash_patt;
```

The polyline texture may be either a line width or line cross-section definition. If the texture is a line width, the width is specified in *attrib*[4]. If the texture is in the form of a line cross-section, the bitmap handle for the line cross-section definition is specified by *attrib*[5].

When only part of the input attribute array is passed in rather than the whole array, the effect will be as if one had called an inquiry of the full array, copied onto that inquiry the part of the array actually passed in, and then invoked the set with the entire array.

Only exactly correct attribute settings are accepted via the attribute array. If a requested attribute setting is not available, the current setting of the attribute is not changed. This behavior differs from the individual attribute setting functions, where defaults and adjustments are available. The intent is that the setting functions be used to set all the attributes associated with a given primitive to values derived from an earlier inquiry.

See the Open Workstation parameters *work_out*[6], *work_out*[7], and *work_out*[13] for device-dependent information regarding this function.

3.127 Set Line Type

Class

Attribute Functions

Inputs

<code>dev_handle</code>	CGIhandle for a currently open workstation
<code>type_in</code>	requested line type

Outputs

None

Value Returned

≥ 0	selected linetype
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Inquire Line Representation
Open Workstation
Output Arc
Output Elliptical Arc
Output Polyline
Set Line Color Index
SetLine Cross Section
SetLine Width
SetLine Representation
SetUserLineType

Description

The Set Line Type function sets the line type (dash pattern) for subsequent polylines and arcs. The total number of line types available is device-dependent. However, the following standard set of pre-defined line types are provided on all devices:

0	user-defined
1	solid
2	long dashed
3	dotted
4	dash-dotted
5	medium dashed
6	dashed with two dots
7	short dashed
>7	device-dependent

If the requested line type is invalid, line type 1 will be used.

See the Open Workstation parameter *work_out*[6] to determine the number of line types available to the device represented by *dev_handle*.

3.128 SetLine Width**Class**

Attribute Functions

Inputs

<code>dev_handle</code>	CGI handle for a currently open workstation
<code>wid_in</code>	requested line width in VDC units

Outputs

None

Value Returned

≥ 0	selected line width
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

InquireLineRepresentation
 Open Workstation
 Output Arc
 Output Elliptical Arc
 Output Polyline
 Set Line Color Index
 Set Line Cross Section
 Set Line Type
 Set Line Representation
 SetUserLineType

Description

The Set Line Width function sets the width for subsequent polylines and arcs.

If the requested line width is outside of the device's capabilities, the line width is set to one device unit and returned in VDC units.

See the Open Workstation parameter *work_out[7]* to determine the number of line widths available to the device represented by *dev_handle*. See the Open Workstation parameters *work_out[62]* and *work_out[63]* to determine the minimum and maximum line widths for the device represented by *dev_handle*.

3.129 SetMarkerColorIndex**Class**

Attribute Functions

Inputs

dev_handle	CGIhandle for a currently open workstation
ind_in	requested marker color index; 0 to the device maximum

Outputs

None

Value Returned

≥ 0	selected color index
-1	error occurred; actual error can be retrieved by invoking Inquire CGIError

Related Functions

Inquire Color Representation
 Inquire Marker Representation
 Open Workstation
 Output Polymarker
 Set Color Representation
 Set Marker Height
 Set Marker Representation
 Set Marker Type

Description

The Set Marker Color Index function sets the color index in which subsequent markers will be displayed.

At least two color indices are provided: foreground and background. Color indices range from zero to a device-dependent maximum.

Use the Set Color Representation function to change the appearance of a color index.

If the color specified is invalid, the closest value in range is chosen. In any case, the selected color index is returned.

See the Open Workstation parameter *work_out*[13] to determine the number of color indices available to the device represented by *dev_handle*.

3.130 SetMarker Height**Class**

Attribute Functions

Inputs

<code>dev_handle</code>	CGIhandle for a currently open workstation
<code>hgt_in</code>	requested marker height in VDC units

Outputs

None

Value Returned

≥ 0	selected height in VDC units
-1	error occurred; actual error can be retrieved by invoking Inquire CGIError

Related Functions

Inquire Marker Representation
 Open Workstation
 Output Polymarker
 Set Marker Color Index
 Set Marker Representation
 Set Marker Type

Description

The Set Marker Height function sets the size of subsequent polymarkers.

If the requested marker height is outside of the capabilities of the device, the marker height is set to the closest device size. If the requested marker height does not exactly map to a device-supported size, then the largest device size that is not greater than the requested marker height is used.

Marker sizes, just like graphics text sizes, are specified by VDC values.


See the Open Workstation parameter `work_out[9]` to determine the number of marker sizes available to the device represented by `dev_handle`. See the Open Workstation parameters `work_out[64]` and `work_out[65]` for the minimum and maximum marker heights.

3.131 SetMarkerRepresentation

Class

Attribute Functions

Inputs

<code>dev_handle</code>	CGIhandle for a currently open workstation
<code>n</code>	number of elements in the array <i>attrib</i>
<code>attrib[0]</code>	requested marker type: <ul style="list-style-type: none"> 0 user-specified; see <i>attrib[3]</i> 1 . 2 + 3 * 4 <input type="checkbox"/> 5 X 6  >6 device-dependent
<code>attrib[1]</code>	marker color index
<code>attrib[2]</code>	marker height in VDC units; used only if <i>attrib[0]</i> is not 0
<code>attrib[3]</code>	handle of bitmap for user-specified marker; valid only if <i>attrib[0]</i> is 0
<code>attrib[4]</code>	x coordinate of the marker hot spot; used only if <i>attrib[0]</i> is 0
<code>attrib[5]</code>	y coordinate of the marker hot spot; used only if <i>attrib[0]</i> is 0

Outputs

None

Value Returned

≥ 0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Inquire Marker Representation
 Open Workstation
 Output Polymarker
 Set Marker Color Index
 Set Marker Height
 Set Marker Type

Description

The Set Marker Representation function uses the specified attributes in subsequent polymarker primitives.

When only part of the input attribute array is passed in rather than the whole array, the effect will be as if one had invoked an inquiry of the full array, copied onto that inquiry the part of the array actually passed in, and then invoked the set with the entire array.

Only exactly correct attribute settings are accepted via the attribute array. If a requested attribute setting is not exactly available, the current setting of the attribute is not changed. This behavior differs from the individual attribute setting functions, where defaults and adjustments are available. The intent is that the setting functions be used to set all the attributes associated with a given primitive to values derived from an earlier inquiry.

Bitmap selection is independent of the marker type. In other words, one can set a bitmap in a given call to Set Marker Representation while not selecting the marker bitmap type, and then at a later time select bitmap type, getting the selected bitmap.

See the Open Workstation parameters `work_out[8]`, `work_out[9]`, and `work_out[13]` for device-specific information regarding this function.

3.132 SetMarkerType

Class

Attribute Functions

Inputs

dev_handle	CGI handle for a currently open workstation
type_in	requested marker type

Outputs

None

Value Returned

≥ 0	selected marker type
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Inquire Marker Representation
Open Workstation
Output Polymarker
Set Marker Color Index
Set Marker Height
Set Marker Representation

Description

The Set Marker Type function sets the marker (symbol) type for subsequent polymarker operations.

The total number of markers available is device-dependent; however, the following six marker types are always provided:

0	user-specified
1	.
2	+
3	*
4	□
5	X
6	◇
>6	device-dependent

If the requested marker type is out of range, type 3 is used.

See the Open Workstation parameter *work_out*[8] to determine the number of marker types available to the device represented by *dev_handle*.

3.133 SetPenSpeed

Class

ControlFunctions

Inputs

dev...handle	CGIhandle for a currently open workstation
speed	requested pen speed as percentage of maximum speed; 1-100

Outputs

None

Value Returned

≥ 0	selected pen speed
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

None

Description

The Set Pen Speed function sets the plotter pen speed to a percentage of the maximum speed between 1 and 100. Requested speeds outside the range will be set to the closest speed within range. This setting affects only plotter devices.

This call can be used to slow down a plotter when using nonstandard inks or media.

The mapping from requested pen speed to actual speed on a particular device is device-dependent.

3.134 SetUserLine Type**Class**

Attribute Functions

Inputs

<code>dev_handle</code>	CGIhandle for a currently open workstation
<code>pattern</code>	32-bit integer to be used as the line dash pattern

Outputs

None

Value Returned

≥ 0	no error
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

Inquire Line Representation
 Output Arc
 Output Elliptical Arc
 Output Polyline
 Set Line Color Index
 Set Line Cross Section
 SetLineRepresentation
 Set Line Type
 Set Line Width

Description

The application may define a line dash pattern (style) and use that definition when drawing polylines. The pattern is defined by the use of a 32-bit integer, where each bit that is on (1) represents when a pixel is turned on and each bit that is off (0) represents when a pixel is turned off. The current writing mode and background mode will be honored.

As an example, the following pseudocode segment sets up a pattern of a long dash, followed by a dot using a hexadecimal constant to define the pattern.

```
longint pattern = FC30FC30hex
:
:
SETUSERLINE TYPE (dev.handle, pattern)
:
:
SETLINE TYPE (dev_handle, 0)
```

The default user line type is solid (FFFFFFFF hex).

3.135 Set Writing Mode**Class****Control Functions****Inputs**

dev_handle

CGIhandle for a currently open workstation

mode_in

requested writing mode

Table 3-2. Writing Modes

1	d = 0 (clear, all bits of color index off)
2	d = d and s
3	d = (not d) and s
4	d = s (replace)
5	d = d and (not s)
6	d = d (no change)
7	d = d xors
8	d = d ors (overstrike)
9	d = not(d ors)
10	d = not(dxors)
11	d = not d
12	d = (not d) ors
13	d = not s
14	d = d or (not s)
15	d = not(d and s)
16	d = 1 (all bits of color index on)

Key: s = color index of source pixel to write
d = color index of destination pixel on display surface

Outputs

None

Value Returned

≥ 0	selected writing mode
-1	error occurred; actual error can be retrieved by invoking <code>Inquire CGIError</code>

Related Functions

CopyBitmap
Inquire Drawing Bitmap
Open Workstation
Output Arc
Output Bar
Output Cell Array
Output Circle
Output Ellipse
Output Elliptical Arc
Output Elliptical Pie Slice
Output Filled Area
Output Graphics Text
Output Pie Slice
Output Polyline
Output Polymarker
Select Drawing Bitmap

Description

The Set Writing Mode function sets the current writing mode. It specifies the boolean operation that is performed between the color indices of the source and destination pixels when graphics such as lines, text, and filled areas are placed on the display.

Default values are device-dependent. Mode 4, replace, is the default for printers and screen devices. Mode 8, overstrike, is the default for plotters. Plotters do not honor writing mode. If an invalid mode is selected, the default is used.

Writing modes will work on the printers in a manner similar to screen device operation. If two objects are displayed on top of each other, the result will be determined by the current writing mode.

See Open Workstation parameter *work_out*[46] to determine the number of writing modes available to the device represented by *dev_handle*.

Note

This function refers only to Graphics Mode. It does not affect cursor text.

3.136 Update Workstation

Class

Control Functions

Inputs

dev_handle CGI handle for a currently open workstation

Outputs

None

Value Returned

0 no error

-1 error occurred; actual error can be retrieved by invoking Inquire CGIError

Related Functions

- Clear Workstation
- Close Workstation
- Inquire Drawing Bitmap
- Open Workstation
- Select Drawing Bitmap

Description

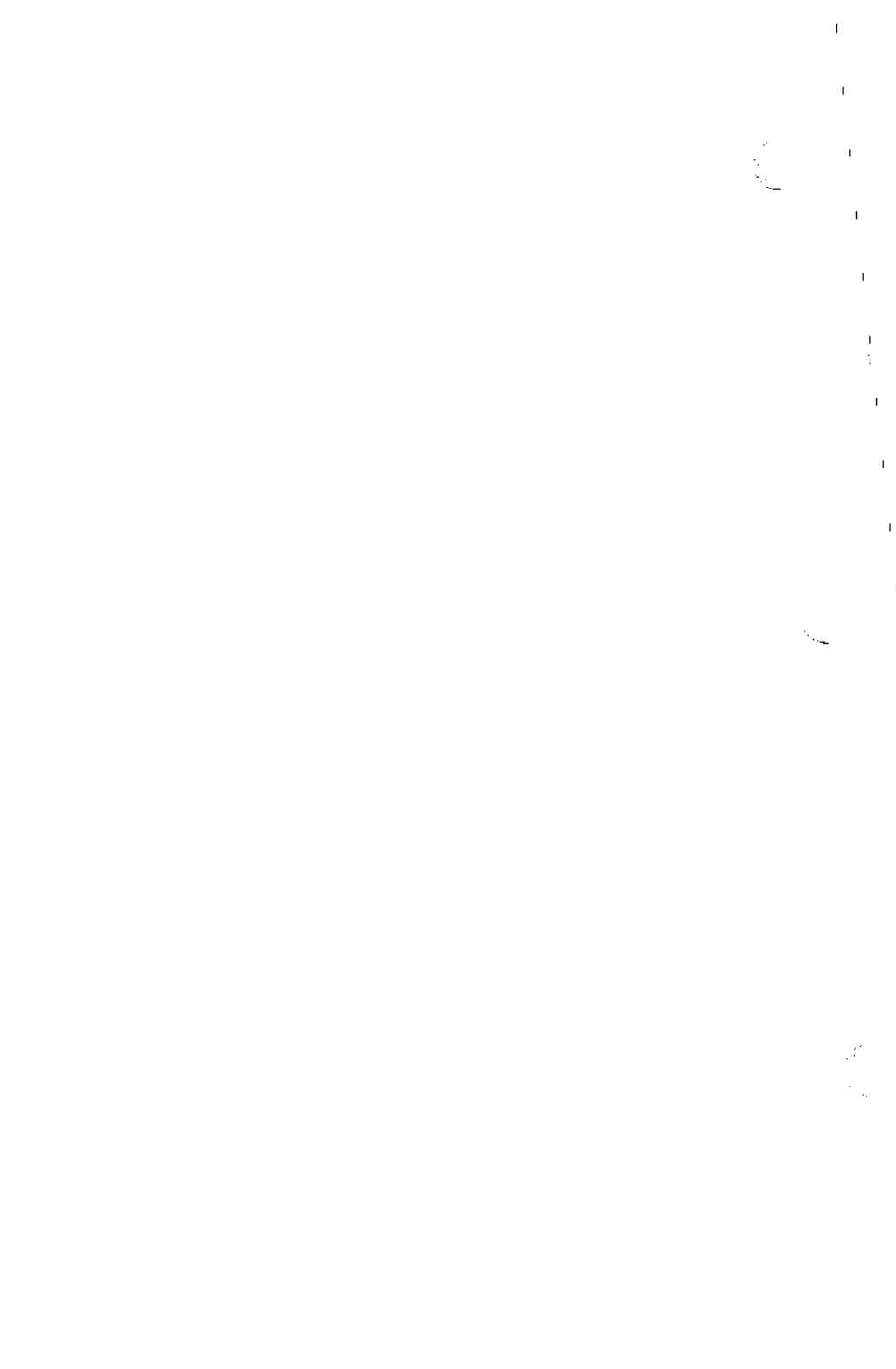
The Update Workstation function displays all pending graphics on the workstation. For printers, this causes the current picture to be output and the printer to be advanced to top-of-form.

Appendix A

Error Codes

A.1 Overview A-1

A.2 ErrorCodeDescriptions A-1



A.1 Overview

This appendix lists the error codes that are returned by SCO CGI functions.

SCO CGI functions always return to the calling program, whether the requested operation was successful or not. Each function returns a value that indicates the results of the request. (These values are listed for each function in Chapter 3 under the heading **Value Returned**.) In this way, the condition of the graphics subsystem is made available to the application program which can then take appropriate action without losing control of the system. This also places the responsibility for checking error status, informing the user, and attempting error recovery, on the application program.

In general, a negative return always implies an error and a return greater than or equal to zero indicates no error.

By default SCO CGI does not display error messages on the system console. Error codes are either system dependent error codes or CGI specific error codes.

The system dependent error codes are the negated value of error codes found in the XENIX Operating System include file `/usr/include/errno.h`.

The CGI specific error codes are referenced by the following list of four digit numbers. Explanations of the CGI specific error codes are also included.

A.2 Error Code Descriptions

Code	Description
-2000	<p>ILLEGAL HANDLE An incorrect device handle was specified. Check the device handle value returned by the Open Workstation routine.</p>
-2001	<p>UNKNOWN DRIVERFILE The device driver file does not exist.</p> <p>Within SCO CGI, device drivers are referred to by logical device names, (e.g., DISPLAY, PRINTER, PLOTTER, etc.). These names must be associated with graphics device drivers.</p>

1. Check that you are requesting the correct device. Refer to the section entitled **Setting Environmental Parameters**, in the release notes. Examine the information about assigning logical device names to graphics devices.
2. Check that you have spelled the device name correctly.
3. Check that the driver you are requesting is located in the VDIPATH specified.

-2002

NO DRIVER FILE

The name of the logical device was not found.

Recommended Action

Ensure that the logical device name is defined in the current shell environment.

-2003

CANNOT START DEVICE DRIVER

The fork command failed to start the graphics device driver as a new process.

Recommended Action

1. Wait until the total number of executing processes is less than the system imposed process limit.
2. Wait until the total number of executing processes for a single user is less than the system imposed process limit.
3. Check to see if there is enough paging space or physical memory for the process. If necessary, increase the paging area or add more physical memory.

-2004

NO MORE DEVICE DRIVERS CAN BE OPENED

The device driver cannot start because the maximum number of device drivers are already opened.

Each application is limited to having no more than eight graphics devices open simultaneously.

Recommended action

Close one of the open graphics devices.

-2005

CANNOT CREATE SEMAPHORE

The device driver cannot start because a system semaphore cannot be acquired.

Recommended action

See the CANNOT ATTACH MEMORY AREA error.

-2006

CANNOT CREATE SHARED MEMORY AREA

The device driver cannot start because a system shared memory area cannot be acquired.

Recommended action

See the CANNOT ATTACH MEMORY AREA error.

-2007

CANNOT ATTACH MEMORY AREA

The device driver cannot start because a system shared memory area cannot be attached to the device driver process.

Recommended action

The previous three error messages occur when the inter-process communication (IPC) of the system will not allow a device driver to start. Some possible corrective measures are:

1. Reduce the size of the shared memory buffer. This is done by changing the value of the SHMMAX environment variable as explained in the release notes.
2. Rebuild the operating system with more IPC areas and reboot your system.
3. Remove any unused shared memory areas with the *ipcrm(1)* command.
4. Terminate any unused running applications.

-2019 **DEVICEBUSYERROR**
The requested channel's physical device is in use and, therefore, not available.

This error is generated when a lock file has been created for the device you are attempting to use. A lock file is created each time you perform graphics to a device that is not assigned to the logical device */dev/tty*. The naming of the lock file has the form:

`/tmp/LCK..ttnn`

Where *nn* is the identification of the logical device being requested.

Recommended action

1. If the lock file was created by another graphics application, then you must wait for that other graphics application to release control of the device.
2. If the file was a remnant of a previous program error, delete the lock file and `run` your program again.

-2020 **OPENDEVICEERROR**
The physical device associated with the requested channel could not be opened.

-2021 **CREATELOCKFILEERROR**
Unable to create the lock file for the requested logical device.

-2022 **UNABLE TO CREATE PROCESS**
The attempt to create a process to associate with the requested channel failed.

-2023 **FILECREATIONERROR**
Attempt to create a file to be used for I/O redirection failed.

-2024 **INVALID DEVICE ASSIGNMENT**
The requested channel has an invalid device assignment.

Recommended action

Check your environmental settings with the *env(1)* system command.

-2025

COULD NOT OPEN FONT FILE

The Font Manager could not open the requested font file.

-2026

FONT DOES NOT EXIST

The requested font index is out of range or the font does not exist.

-2027

FONT FILE MISSING

The Font Manager could not find the requested font file. This can indicate that the font database file is out of date.

Recommended action

Run the font utility program *instfont* to update the font database file.

-2028

MISSING FONT MANAGER DATABASE FILE

The Font Manager could not find the database file *fontlist.dat*.

Recommended action

Check your environmental settings with the *env(1)* system command. If the font path is correctly set, and if font data files exist but the font database file *fontlist.dat* does not exist, execute the font utility program *instfont* to create the database file. See the release notes for instructions on how to set up your font path.

-2029

READ FONT FILE ERROR

The Font Manager encountered an error while reading the font file.

-2977

CGI ALREADY LOADED

This error is returned from Load CGI and should be ignored. It is there for compatibility with DOS applications.

-3000

DEVICE DRIVER ERROR

An error occurred when calling the device driver.

- 3001** **DEVICEDRIVERINVALIDRANGEERROR**
A parameter that was passed from the application was out of the valid range for the operation.
- Reconunended action**
- Check the routine description in Chapter 3, **SCO CGI Functions**, and the appropriate language reference booklet for the correct parameter definitions.
- 3002** **HANDLEINUSEERROR**
The bitmap handle is in use and can not be deleted.
- 3004** **INCOMPATIBILITYERROR**
The device driver and the CGI controller are incompatible.
- 3005** **TOOBIGBITMAPERROR**
The requested bitmap is too large.
- 3008** **DEVICEDRIVERCOULDNOTMALLOC**
There is insufficient memory to create the intended bitmap.
- 3090** **INVALIDPAGEERROR**
The physical page is invalid.
- 3095** **DEVICENOTINCURSORMODE**
The device is not in cursor text mode.
- 5000** **DEVICEDRIVERNOTCAPABLEERROR**
The specified device is not capable of performing the requested function.

Appendix B

SCO CGI Architecture

B.1 Overview B-1

B.2 Device Drivers B-1

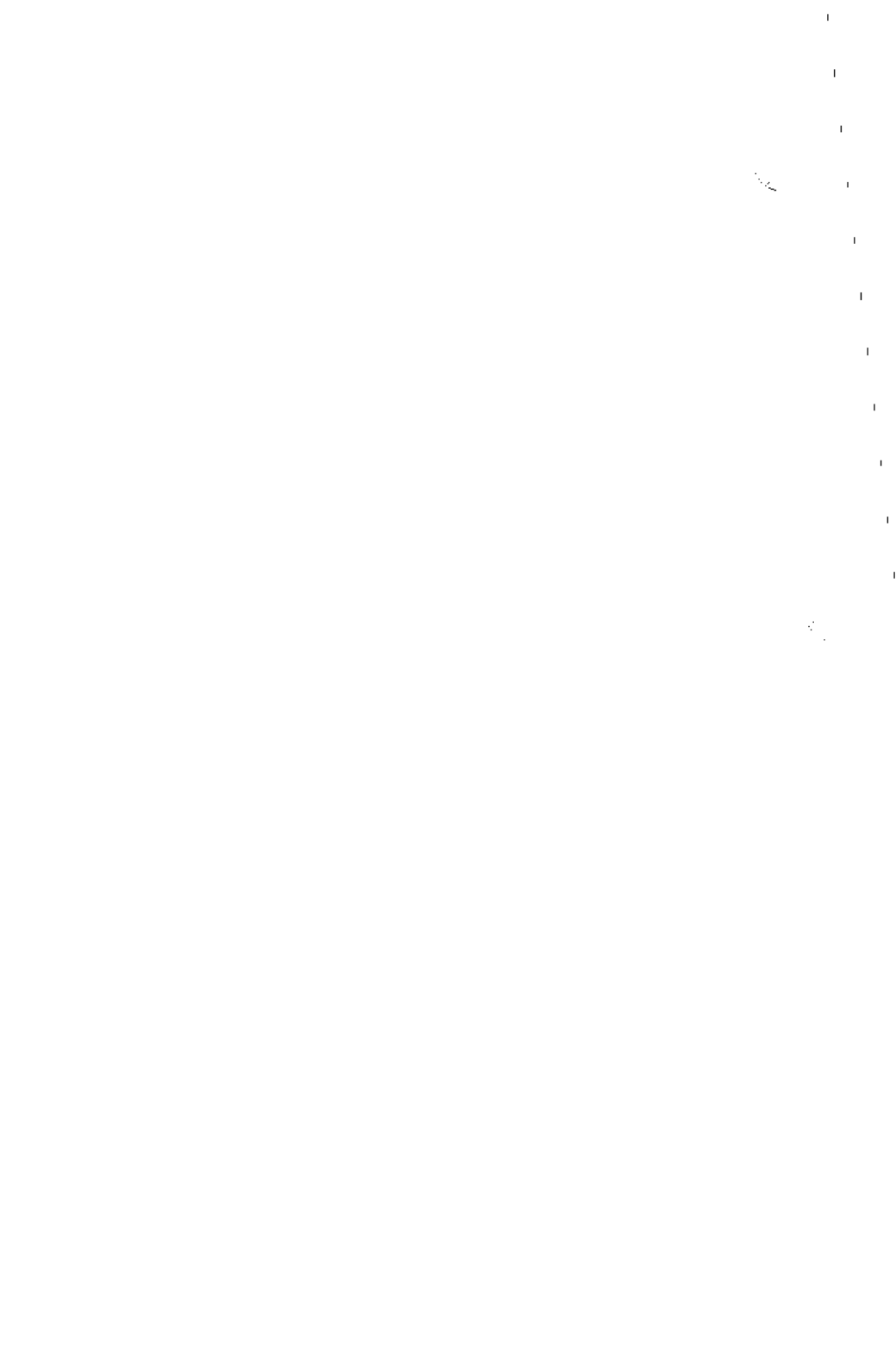
B.3 BindingPart B-1

B.3.1 Coordinate Transformation B-1

B.3.2 Device-Driver Management B-1

B.3.3 Error Handling B-2

B.3.4 Graphics Emulation B-3



B.1 Overview

SCO CGI has two major components:

- a set of device drivers that form the interface between the CGI and specific graphics peripheral devices; and
- the binding part of the graphics application that provides a computer-independent environment for graphics.

B.2 Device Drivers

The device drivers are provided as executable modules that can be called from an application program, to convert device-independent data to device-specific data needed to operate graphics hardware.

B.3 Binding Part

The binding portion of the system provides an environment for the creation of graphics applications that can be transported to any system conforming to the graphics standards.

B.3.1 Coordinate Transformation

Graphics information is passed to SCO CGI in virtual device coordinates (VDC [-32768...+32767]) that are independent of any particular device. The system uses information obtained from the device driver to scale the virtual coordinates to device coordinates that are consistent with the values used by a particular graphics device (for example, raster steps). It also transforms input device coordinates into virtual coordinates.

B.3.2 Device-Driver Management

Each physical graphics workstation may have several devices attached. In addition, SCO CGI allows applications to be device-independent so that the devices can be interchanged if the need arises. In order to conserve resource requirements, only currently requested device drivers are brought into memory. This dynamic loading of device drivers means that the user need not be aware of the changes.

The SCO CGI Open Workstation routine loads the driver from disk storage before performing the graphics operation.

SCO CGI also incorporates a metafile driver that generates metafiles consistent with the proposed ANSICGM standard. This capability is invoked by assigning METAFIL as the output device. The metafile device driver creates a metafile that is saved on the system storage device.

Metafiles may be read using a separate SCO product called (see the *Programmer's Guide* for more information).

The device driver file names must be unique, but there are no other constraints. For example, the device driver file name for a Hewlett-Packard Thinkjet printer is: thinkjet

B.3.3 Error Handling

The condition of the graphics subsystem is made available to the application program which can take appropriate action without losing control of the system. It is the responsibility of the application program to check error status, inform the user, and attempt error recovery.

Open Workstation and all other functions also return a value (error state) that indicates whether the requested command was completed successfully. If a negative value is returned after issuing a command when using SCO CGI high-level language bindings, invoke Inquire CGI Error to identify the specific error, or invoke Output CGI Error to print the associated error message. Messages associated with each error value are listed in Appendix A, **Error Codes**.

Error code values are derived according to the following rules.

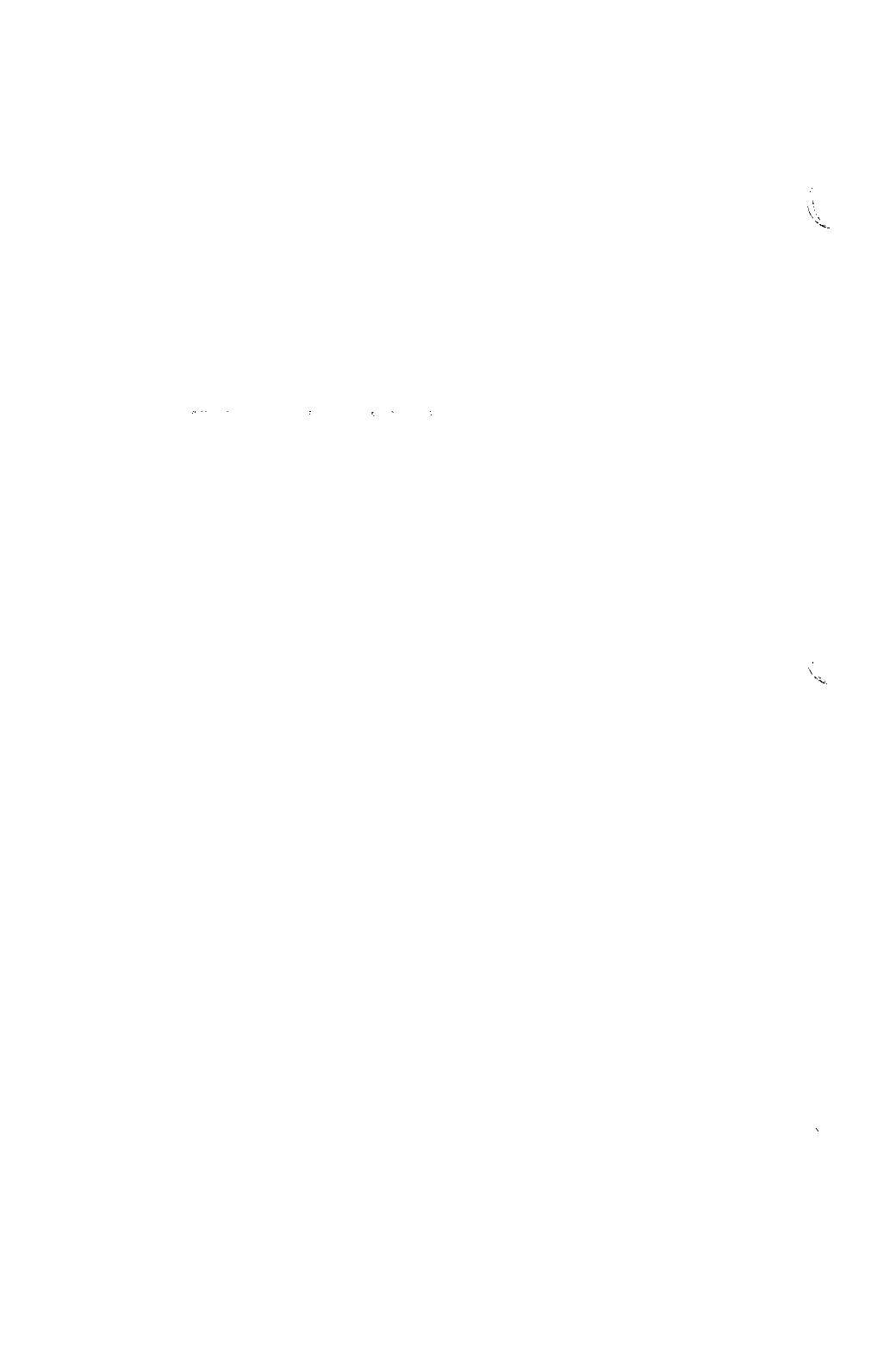
- A negative code indicates an error.
- A code greater than or equal to zero indicates that the function call completed successfully.
- Error codes are either system dependent error codes or CGI specific error codes.

Device drivers are designed to perform the operations requested by the application to the best of the device's ability. However, some devices are not capable of performing certain operations. For instance, a vector device such as a plotter cannot perform bitmap operations. In such a case, the device driver returns to the application an error indicating that the device driver is not capable of performing that operation. Any errors prevent the driver from completing the requested operation.

Cursor Addressing Mode has no effect on printers or plotters. It does not return errors; instead, the functions are ignored. Inquire Addressable Character Cells returns an error message indicating that cursor text is not available.

B.3.4 Graphics Emulation

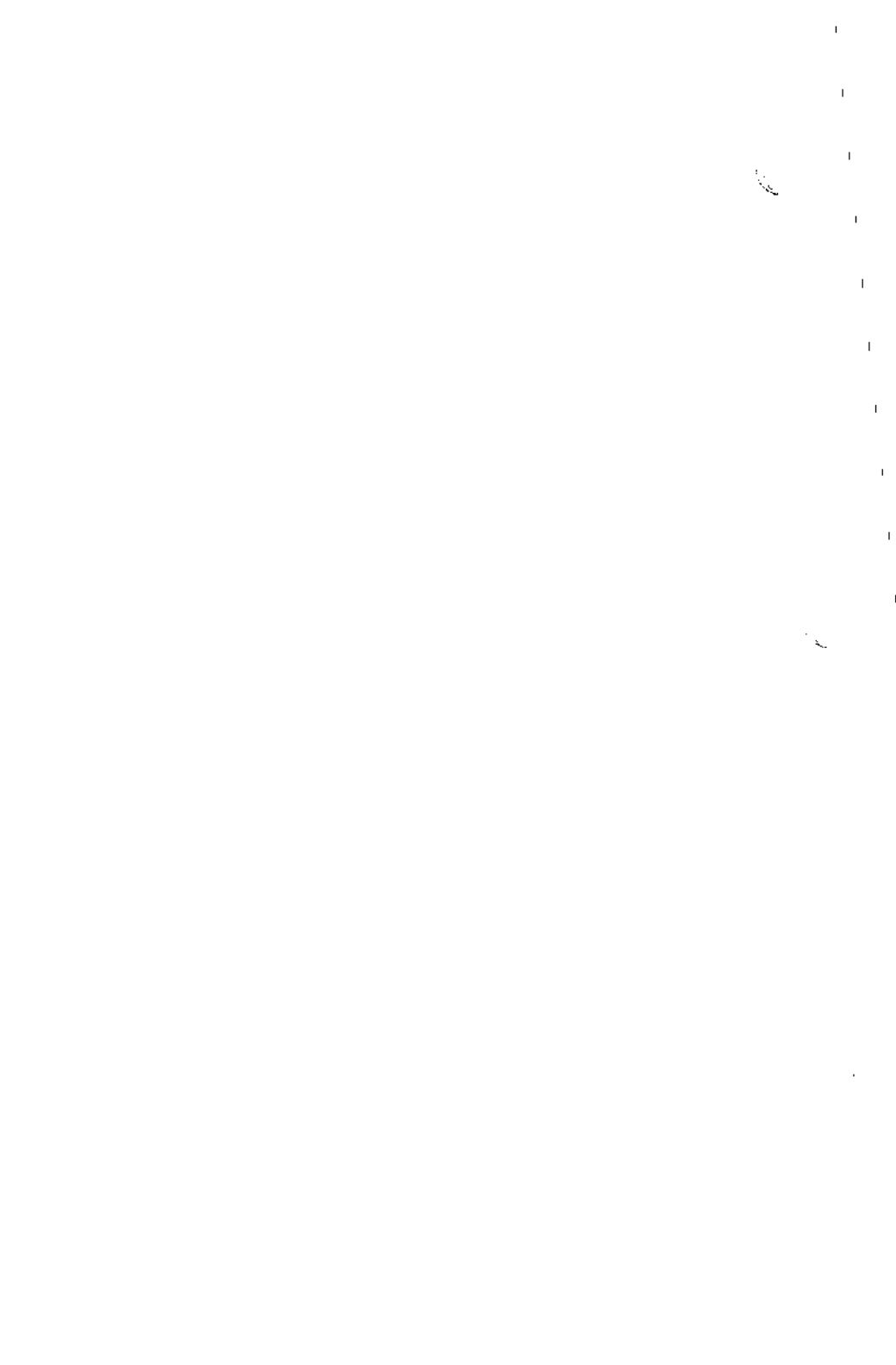
The CGI incorporates many sophisticated graphics notions. In many cases the capabilities offered by a graphics device are only a subset of the total CGI possibilities. In order to ensure application portability, some functions that are not supported directly by the device are emulated by SCO CGI. Refer to the *SCO CGI Device Driver Supplement* for device-specific information.



Appendix C

Font File Architecture

- C.1 Raster Font File Architecture C-1
- C.2 FontFileFormat C-1
 - C.2.1 Arrangement of Data in Font File C-2
 - C.2.2 Constants C-2
 - C.2.3 Font File Header C-3
 - C.2.4 Font Header C-3
 - C.2.5 Font Metrics C-6
 - C.2.6 Character Descriptor C-8
 - C.2.7 Kerned Characters C-10
 - C.2.8 Character Positioning C-11
 - C.2.9 Accessing Font File Information C-12
 - C.2.10 Accessing the TXFONTHEADER structure C-12
 - C.2.11 Accessing a FONT_METRICS structure C-12
 - C.2.12 Accessing a TXBITMAPCHAR structure and character bitmap C-12



C.1 Raster Font File Architecture

This description provides applications developers and users the capability to construct additional fonts that can be accessed and displayed through an SCO device driver, thus allowing a wide range of character sets, font faces, and sizes.

The font information is presented as a list of C structure declarations, followed by a description of the structure in English. The font file contents must support the system dependencies associated with word alignment of data within a C structure. This can be done by doing read or write operations on the structure, rather than on the individual fields in the structure. SCO-compatible font files are such that the appropriate C structures can be read from disk in one operation, without manipulating the bits afterwards.

C.2 Font File Format

A raster character font is defined as a collection of bit images that make up the individual characters of the font. The characters can be of unequal widths to support proportional character spacing. Kerning is achieved by moving the second of appropriate pairs of characters to the left; for example, the descender of the lowercase *j* can extend under certain previous characters. A font can contain up to 256 characters, but not all of the characters need be defined. Each font contains a symbol that is drawn in place of a request for a missing font character.

Text is drawn in various sizes. Three techniques exist for obtaining different sized characters:

1. supply bitmap descriptions for multiple sizes of characters;
2. allow the characters to be scaled from a single description by pixel replication; and,
3. allow continuous scaling by multiplying the coordinates of the character description by a constant factor.

Device drivers currently available support only the first option.

Most bitmap fonts are scaled using the first technique. The font file contains bitmap descriptions of each character in several discrete sizes. This allows the optimum character design for each size – an important feature when using low-resolution devices.

C.2.1 Arrangement of Data in Font File

The format of the SCO font file is as follows:

1. One TXFILEHEADER structure.
2. One TXFONTHEADER structure.
3. For each text size:
 - One FONT_METRICS structure.
 - One set of long integer-size offsets, each offset pointing to the start of a character descriptor in the font file.
 - One long integer-size area that is reserved for future use.
 - A set of character descriptors, one for each character in the font.

Each character descriptor includes:

- One TXTBITMAPCHAR structure.
- The bitmap of the character.

C.2.2 Constants

Several of the structures below reference named constants. The values of these constants are listed here.

```
#define MAX_FONT_NAME 20    /* maximum length of a fontname */

#define SW_BITMAP 0        /* font type is Software Bitmap */
#define HW_BITMAP 1        /* Hardware Bitmap */
#define SW_OUTLINE 2       /* Software Outline */
#define HW_OUTLINE 3       /* Hardware Outline */

#define MULTIPLE_SIZES 0   /* if multiple sizes are available */
#define REPLICABLE 1      /* if the font is pixel replicable */
#define SCALEABLE 2       /* if the font is scaleable */

#define ASCII 0            /* if the font is X3.4 ASCII */
#define ANSI 1             /* if the font is ANSI */
#define SPECIAL 2         /* otherwise it is special */
```

```
#define MONO_SPACED 0      /* if the font is mono-spaced */
#define PROPORTIONAL 1    /* if the font is proportionally spaced */
#define NOT_KERNED 0      /* if kerning information is not available */
#define KERNED 1          /* if kerning information is available */

#define UNROTATED 0       /* font contains only 0° rotation */
#define ROTATED 1         /* only 90° rotation characters here */
#define BOTH_ROTATIONS 2  /* both 0° and 90° data available */
```

C.2.3 Font File Header

The font file header provides the user with information about the contents of a file. This information is not useful to the program using the font, but identifies the font file for the user. After the identifying information, there is a zero byte followed by a CTRL-Z to terminate the file description.

```
typedef struct
{
  char font_desc[128];      /* English description of font */
  char copyright[126];     /* Copyright notice */
  char null;               /* EOS mark */
  char CTRL_Z;             /* EOF mark for some systems */
} TXFILEHEADER;
```

font_desc:	128 characters of text This is a readable version of the contents of the font file. It is seen when the file contents are dumped, but has no effect on computer programs. It usually contains a carriage return at the end so that the copyright notice will appear on a new line.
copyright:	126 characters of text This contains the name of any copyright holder on the file contents.
null:	one character containing a zero If the header is read and displayed as a string in C, this will terminate the string.
CTRL_Z	one character containing a CTRL-Z character (decimal 26)

C.2.4 Font Header

The font header contains information necessary to display the characters from the font. This information describes the characteristics of the font, independent of its size. It is of little importance to the application user, but is useful to the programmer for general information about the fonts.

SCO CGI Programmer's Guide

```
typedef struct {
    int font_version;           /* SCO font file version */
    char font_type;            /* SW_BITMAP,
                               HW_BITMAP, ... */
    char nom_char;             /* the "nominal" character */
    char spacing_type;         /* MONO_SPACED,
                               PROPORTIONAL */
    char character_set;        /* 0 */
    char font_start;           /* first character in font */
    char font_end;            /* last character in font */
    char undefined_char;       /* what to substitute for missing
                               chars */
    char num_sizes;           /* number of discrete sizes */
    char size_info;           /* REPLICABLE SCALEABLE
                               MULTIPLE_SIZES */
    char rotations;           /* UNROTATED, ROTATED,
                               BOTH_ROTATIONS */

    int nom_horiz;            /* nominal pixel size
                               (micrometers) */
    int nom_vert;             /* " " */
    char kerned;              /* 0 no kerning, 1 kerned */
    char font_name[MAX_FONT_NAME]; /* fontname */
    FONT_METRICS *metrics;
} TXFONTHEADER;
```

- font_version:** an integer-size positive integer
This is the SCO file format version number. This documentation describes Version 2, so any file you build should have a binary 2 here.
- font_type:** a char-size non-negative integer
Because this documentation describes software bitmap fonts, this structure member should always contain the constant 0 (SW_BITMAP).
- nom_char:** a single char
This is the character around which the font is designed. Most font designers use the capital M as the basis for the size of the other characters.
- spacing_type:** a char-size flag
If this value is 0, the font system assumes that all characters in the file are the same width. This allows system efficiency during the display of characters from this font.

- character_set:** a char-size flag
As of Version 2 of the font file, no drivers look at this flag. It should be set to zero for future compatibility.
- font_start:** a char which represents the lowest-valued character defined in this file
For example, a decimal 32 in this position indicates that the first defined character is the ASCII *space* character, and that no ASCII control characters were defined.
- font_end:** a character representing the last defined character in the font
A decimal 127 indicates that ASCII *rubout* was the last defined character.
- undefined_char:** a single character declaring the character to supply in place of undefined entries in the character table
When the font system encounters a character that is not defined, a character chosen by the font designer is substituted. The character is usually the *space* or ? A character is considered undefined if it lies outside the range specified by *font_start* and *font_end*, or if its description contains zero bytes (indicated by two adjacent values in the array of byte offsets being identical).
- num_sizes:** a char-size positive integer
This value indicates the number of separate sizes of characters that are defined in the file. *num_sizes* determines the number of FONT_METRICS structures in the file, and the number of byte offset arrays pointing to character descriptions.
- size_info:** a char-size integer
This value describes the technique used to change sizes in this file. If the value is zero (MULTIPLE_SIZES), there is a separate bitmap for each discrete character size. If the value is one (REPLICABLE), the font is designed to use pixel replication of the base character description to create different sizes. If the value is two (SCALEABLE), the font description is such that continuous scaling can be performed on the font. As of Version 2 of the font file, the only supported value for this parameter is MULTIPLE_SIZES (zero).
- rotations:** a char-size flag
This value is zero (UNROTATED) if the font contains only characters meant to be displayed at a font rotation angle of 0 (or 180) degrees. A value of one (ROTATED) indicates the file contains only characters suitable for display at 90 or 270 degrees. BOTH_ROTATIONS (two) means that character descriptions for both orientations exist. This flag is used in fonts designed for devices whose pixels are non-square.

nom_horiz, nom_vert: two positive integers
 These describe the horizontal and vertical pixel size (in micrometers) of the target device. Device drivers whose pixel sizes do not match these values within an acceptable tolerance (device-specific) will not try to display characters from this font file.

kerned: an char-size flag
 This flag's value is one if the font contains any characters whose descriptions exceed the cell boundary, for example, italic fonts. Device drivers require more time to display fonts designed with kerning.

font_name: a char array of size MAX_FONT_NAME
 This array holds the name of the font.

metrics: a pointer-size location
 This should be set to zero in the font file. It is a place holder and receives a value when the file is read into memory.

C.2.5 Font Metrics

Each character size for a font has a block of information describing the general dimensional characteristics of the font in this size.

```
typedef struct{
  int char_height;      /* nominal character height */
  int char_width;      /* nominal character width */
  int cell_height;     /* nominal cell height */
  int max_height;     /* maximum cell height */
  int cell_width;     /* nominal cell width */
  int max_width;      /* maximum cell width */
  int baseline_offset; /* font baseline offset */
  int body_offset;    /* distance from left cell edge to char rect */
  int space_width;    /* font space width for this size */
  int half_line;     /* halfway between base and cap line */
  char rotated;      /* UNROTATED, ROTATED,
                     BOTH_ROTATIONS */

  int *width_table;
  int *kern_table;
} FONT_METRICS;
```

char_height: an integer
 This describes the design height of a character, measured between the base line and the cap line.

char_width:	an integer The typical width of a character in this size. Usually, this value is the width of the nominal character from the TXFONT-HEADER structure.
cell_height:	an integer This value describes the total designed height of character cells in this font at this size, including the distance necessary to properly space this line of text from the ones above and below.
max_height:	an integer This value describes the maximum height attained by any character cell in the font.
cell_width:	an integer This value describes the typical cell width. It contains the character width, plus any extra space needed on the sides to properly position adjacent characters.
max_width:	an integer This integer contains the maximum of all cell widths in the font at this size.
baseline_offset:	an integer. This value describes the offset from the bottom of the cell to the character baseline. (See Figure 3-14, <i>The Font Coordinate System</i> for an illustration.)
body_offset:	an integer This value gives the offset from the left edge of the character cell to the beginning of the character body. For proportionally spaced fonts, this value is the minimum of those given in the individual characters. A negative value here means that a character in the font exceeds the cell edge to the left (is <i>kerned</i>). (See Figure 3-14, <i>The Font Coordinate System</i> in Chapter 3 for an illustration.)
space_width:	an integer This value gives the width of the space character in this size.
half_line:	an integer This value describes the distance from the base line to the half line of the font. Usually, this value points to the position at which the center line of the B and E characters are located. It need not be exactly half the distance between the base and cap lines.
rotated:	a char-size flag This flag determines whether the set of characters is to be treated as a rotated or non-rotated size. It is possible to have two

FONT_METRICS structures describing characters of identical *char_height*, one of which has *rotated* = UNROTATED and another ROTATED. It is also possible to describe some sizes with *rotate d* = BOTH_ROTATIONS, and some with separate bitmaps for the two orientations. Be sure to have at least one size for each rotation if the FONT_HEADER allows both rotations.

width_table, *kern_table*:

two integer pointers

These two values exist for future compatibility, and are not used for fonts whose *font_version* is 2 (see **Font Header Structure** above.)

C.2.6 Character Descriptor

Each character descriptor contains complete information to display a character. The information is as follows:

```
typedef struct {
    char char_id;           /* which character is this */
    int horiz_offset;      /* where does the character start */
    int char_width;        /* size of the actual character */
    int char_height;       /* " " " */
    int cell_width;        /* size including spacing */
    int cell_height;       /* " " " */
    unsigned num_bytes;    /* (width+7)/8*height */
    char *bits;           /* filled by user to point to bitmap */
} TXBITMAPCHAR;

char bitmap[num_bytes]; /* the bitmap */
```

char_id: a char containing the character described by this structure
This value is used for consistency checking, to be sure that the array of byte offsets actually points to the correct TXBITMAP-CHAR entry in the file.

horiz_offset: an integer
The distance between the left-hand edge of the character cell and the body of the character. For kerned characters, this value may be negative. In that case, the bitmap description of the character starts at the character edge, not the cell edge. (See *body offset* in Figure 3-14, *The Font Coordinate System* in Chapter 3 for an illustration.)

char_width: a non-negative integer
This value describes the number of pixels across the character body. If the sum of this number and the *horiz_offset* exceeds

the *cell_width*, the bitmap width is increased on the right to hold the extra pixels. (See Figure 3-14, *The Font Coordinate System* in Chapter 3 for an illustration.)

- char_height:** a non-negative integer describing the height of this character
- cell_width:** a non-negative integer
This indicates the width of the character cell the character is displayed in, including extra space at the sides if needed.
- cell_height:** a non-negative integer
This indicates the height of the character cell.
- num_bytes:** an unsigned integer
This value (between 0 and 65535) gives the number of bytes needed to describe the character. Its value is usually $(cell_width-7)/8*cell_height$, if the character body does not overlap the cell. If there is any overlap, the extra pixels are added to the width or height before this formula is applied.
- bits:** a pointer to a character bitmap
This pointer must be filled in with the address of the array that the character bitmap will be read into.
- bitmap:** an array of char values
This array is the bitmap description. This is not a strictly legal declaration in C. The size of this array of bytes depends on *num_bytes*, and can vary from character to character. The most significant bit of the first byte in this array is the top left corner of the character cell. Successive bytes describe the top row of the character cell. Each new row starts on an even byte boundary; extra bits at the end of each character row are ignored and should be zero.

The following is an example of a character descriptor for the character A :

```
char_id = 65
horiz_offset = 1
char_width = 10
char_height = 7
cell_width = 12
cell_height = 9
Num_bytes = (12+7)/8*9 = 18

bits={0x00, 0x00,    00000000 0000****
      0x06, 0x00,    00000110 0000****
      0x0F, 0x00,    00001111 0000****
      0x19, 0x80,    00011001 1000****
      0x19, 0x80,    00011001 1000****
      0x3F, 0xC0,    00111111 1100****
      0x30, 0xC0,    00110000 1100****
      0x79, 0xE0,    01111001 1110****
      0x00, 0x00}   00000000 0000****
```

In the above example, the last four bits of each character row are unused.

C.2.7 Kerned Characters

Characters can be kerned in some raster fonts. This allows the character cell of one character to overlap the character cell of another character. This happens in italic fonts. The following explains the structure for kerned characters.

```
typedef struct{
RECT bounds;          /* relative to current set-point */
XY setPoint;         /* to the new set-point */
XY sz;               /* number of bits in both directions */
int rowStep;        /* #bytes between rows */
char *bits;         /* the actual bitmap */
} KERNED_CHAR;
```

bounds: a RECT structure
This structure of two XY structures defines the lower left and upper right bitmap bounds relative to the current set point.

setPoint:	an XY structure This structure of two signed integers defines the delta in x and y from the current setpoint to the new setpoint.
sz:	an XY structure This structure of two signed integers defines the size in x and y of the bitmap.
rowStep:	a signed integer This signed integer is the number of bytes contained in one row of the bitmap.
*bits:	a char pointer This array is the bitmap description. The size of this array is the $((sz.x + 7) / 8) * sz.y$ bytes and can vary from character to character. The most significant bit of the first byte in this array is the top left corner of the character cell. Successive bytes describe the top row of the character cell. Each new row starts on an even byte boundary. Extra bits at the end of each character row are ignored and should be zero.

C.2.8 Character Positioning

The character cell is defined so that character spacing can be done as simply as possible. To achieve proper inter-character spacing the left side of a character cell will be placed adjoining the right side of the previous character cell. The following is an example of placing a B next to an A.

```

Before spacing
000000000000**** 000000000000****
000001100000**** 011111110000****
000011110000**** 001100011000****
000110011000**** 001100011000****
000110011000**** 001111111000****
001111111100**** 001100001100****
001100001100**** 001100001100****
011110011110**** 011111111000****
000000000000**** 000000000000****

```

```

Afterspacing
0000011000000011111110000
000011110000001100011000
000110011000001100011000
000110011000001111111000
001111111100001100001100
001100001100001100001100
01111001111001111111000
000000000000000000000000

```

C.2.9 Accessing FontFileInformation

This section describes how to get information out of a font file. The *sizeof* function is assumed to accommodate padding used by your system's C compiler.

C.2.10 Accessing the TXFONTHEADER structure

This structure is located immediately after the TXFILEHEADER structure at the top of the font file. To access this structure:

1. Seek to byte `sizeof(TXFILEHEADER)`.
2. Read `sizeof(TXFONTHEADER)` bytes into your structure.

C.2.11 Accessing a FONT_METRICS structure

There is one FONT_METRICS structure for each character size in the file. To access the third structure, first set *size* = 2. Then:

1. Set *start* = `sizeof(TXFILEHEADER) + sizeof(TXFONTHEADER)`.
2. Seek to byte *start* + `sizeof(FONT_METRICS) * size`.
3. Read `sizeof(FONT_METRICS)` bytes into your structure.

C.2.12 Accessing a TXBITMAPCHAR structure and character bitmap

For each character size, there is an array of long integer offset values giving the location of each character in the file. The size of each array can be calculated using two values, *font_start* and *font_end*, in the TXFONTHEADER structure. The location of the first byte offset array is immediately after the last FONT_METRICS structure, and TXFONTHEADER element *num_sizes* gives the number of FONT_METRICS structures. If *num_sizes* = 3, to access the character description for the character *ch*, follow this procedure:

1. Set *start* = `sizeof(TXFILEHEADER) + sizeof(TXFONTHEADER)`.
2. Add `sizeof(FONT_METRICS) * num_sizes` to *start*.
3. Set *tablesize* = `sizeof(long) * (font_end - font_start + 2)`.
4. Seek to byte *start* + (*tablesize* * (*num_sizes* - 1)) in file.

5. Read *tablesize* bytes from file into an array of long integers. Call this array *char_starts*. (This array is valid until the next time the character size changes.)
6. Set *start* = *char_starts* [*ch* - *font_start*].
7. Set *end* = *char_starts* [*ch* - *font_start* + 1].
8. Seek to byte *start* in the file.
9. Read *sizeof*(TXBITMAPCHAR) bytes into a TXBITMAPCHAR structure. Read *num_bytes* bytes into a bitmap array. Make sure the bitmap array is large enough for the character you are reading. The *char_id* element of this structure should match *ch*, or there is a consistency problem in the file.

If (*end* - *start*) is zero, there is no bitmap defined for this character. The font system will search for the *undefined_char* in this case.

2

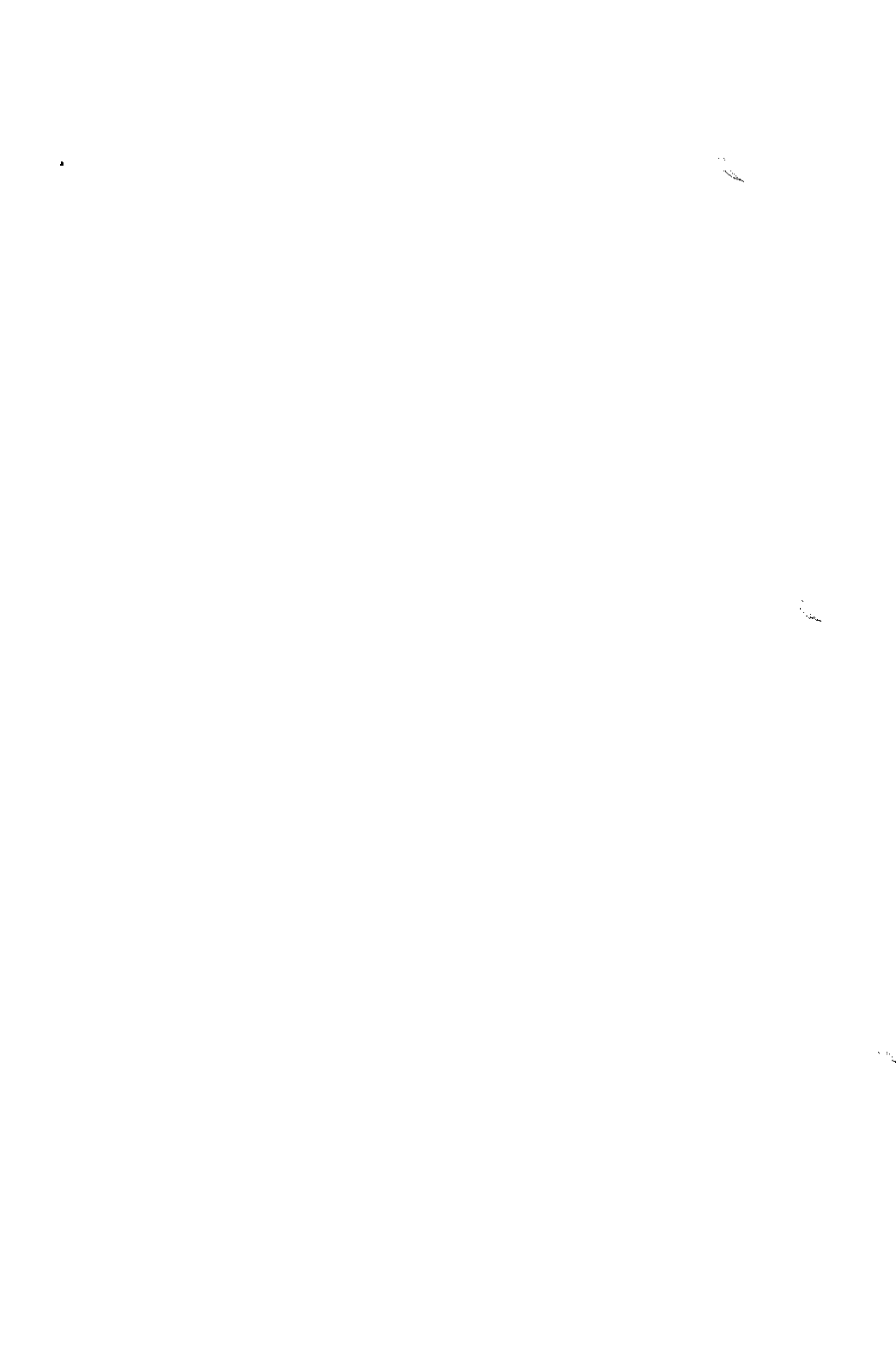
.

2

Appendix D

ASCII Code Chart

D.1 Overview D-1



D.1 Overview

The following chart shows the ASCII character codes that are used throughout the computer industry.

CONTROL		NUMBERS SYMBOLS	UPPER CASE		LOWERCASE	
NUL	DLE	SP	0	@	P	p
SOH	DC1	!	1	A	Q	q
STX	DC2	"	2	B	R	r
ETX	DC3	#	3	C	S	s
EOT	DC4	\$	4	D	T	t
ENQ	NAK	%	5	E	U	u
ACK	SYN	&	6	F	V	v
BEL	ETB	'	7	G	W	w
BS	CAN	(8	H	X	x
HT	EM)	9	I	Y	y
LF	SUB	*	:	J	Z	z
VT	ESC	+	;	K	[k
FF	FS	,	<	L	\	l
CR	GS	-	=	M]	m
SO	RS	.	>	N	^	n
SI	US	/	?	O	_	o

Figure E-1. ASCII Code Chart

2

3

Glossary

- ADE** See *ASCII Decimal Equivalent*.
- alpha text** Text which can be produced in several discrete sizes, placed anywhere on the display surface, and which supports several fonts. Alpha text is displayed in graphics mode. Compare *cursor text*, *graphics text*.
- argument** One of the independent variables on which the action or output of a routine depends. Arguments are enclosed in parentheses in the routine call.
- ASCII** American Standard Code for Information Interchange. This standard for data transmission assigns individual 7-bit codes to represent each of a specific set of 128 numerals, letters, and control characters.
- ASCII Decimal Equivalent** A decimal number used in code to represent an ASCII character. For example, the integer 65 equals the letter *A*, and the integer 66 equals the letter *B*. ADE character parameters are passed and returned as integers.
- aspects of primitives** Ways in which the appearance of a primitive can vary. Aspects are directly controlled by primitive attributes.
- attribute functions** Functions which alter the appearance of graphics objects created with graphics primitive functions. For example, line style is an attribute function which affects the appearance of a line, which is a graphics primitive.
- attributes** Properties that apply to a display element or a segment (such as highlighting, color, or character height).
- background color** The color that determines the backdrop of a chart or display surface. Compare *foreground color*.
- baseline** The imaginary horizontal line upon which all the characters in a given line stand. A descender passes below this line. All baselines of a font are in the same position within the character bodies. See *character body*.

- binding** *See language binding.*
- BitBlt** *A bit boundary block transfer; the act of transferring a rectangular section of pixels from a source to a destination bitmap, while in any given writing mode. See *bitmap*, *writing mode*.*
- bitmap** *A two-dimensional array in memory in which each element of the array represents a pixel. A bitmap is used to draw or manipulate graphics primitives. A raster output device may display the contents of a bitmap.*
- bitmap font** *A text font in which the character is defined as a raster bitmap. Compare *outline font*. See *bitmap*.*
- byte pixel array** *See *pixel array*.*
- calligraphic graphics** *Graphics composed of lines. Also called *vector graphics*.*
- canonical rectangle** *On most devices, a pixel is larger than a virtual device coordinate. When a rectangle is specified in VDCs, therefore, the actual location of its corners might be mid-pixel. The *canonical rectangle* is the rectangle specified by mapping the VDC coordinates of its lower left corner to those of the lower left corner of the pixel in which it falls, and by mapping the VDC coordinates of its upper right corner to those of the upper right corner of the pixel in which it falls. See *Virtual Device Coordinate*, *pixel*.*
- capline** *An imaginary horizontal line within a character body which, for many character definitions, has the appearance of being the upper limit of the character shape, usually at the height of a capital letter. An ascender or an accent mark may pass above this line. All caplines in a font are in the same position within the character bodies. See *character body*.*

Cartesian coordinate system

A coordinate system composed of an x axis (horizontal) increasing positively towards the right, and a y axis (vertical) increasing positively upwards. The axes are positioned at right angles, and the point of intersection is the origin (0,0). The position of any point is defined by the

- displacement from the origin along the *x* and the *y* axes.
- cell array An output primitive consisting of a rectangular grid of equally large rectangular cells, each having a specified color. These cells may not map one-to-one with framebuffer pixels.
- centerline A vertical line bisecting the character body. See *character body*.
- CGI See *Computer Graphics Interface*.
- character body A rectangle used by a font designer to define a character shape. All character bodies in a font have the same height.
- character up vector The vector giving the *up* direction of a character. Only the direction, not the length, of the vector is relevant. The text path is defined relative to the *character up vector*. For Text Right, the text is written along a baseline whose direction is 90 degrees clockwise from the direction of the character up vector.
- choice input device A logical input device that offers the user a set of alternatives and returns an integer value indicating the option selected. An example device is a set of function keys.
- clip indicator An indicator which shows whether graphics elements are to be clipped at the limits of Clip Rectangle. See *clip rectangle*, *clipping*.
- clip rectangle A rectangle defined in VDC space used as a boundary to specify where graphics elements are to be clipped. See *clipping*.
- clipping If the visible space is smaller than the virtual coordinate space, the software must make those portions of the object outside the visible space invisible. The coordinates of the clipping rectangle determine which portion of the virtual coordinate space is visible. If the coordinates outside the visible space were not clipped, the image displayed would vary from one device to another.
- color index An integer identifying a color. The value of a particular color index depends on the device

used. For example, color index 0 may indicate black for CRTs and white for plotters. See *color map*.

- color map** A table designed to provide a range of colors by defining different mixtures of the RGB color components. A desired color is referenced by its assigned number. The identifying numbers with their assigned colors are called the color map. Changing colors assigned to the identifying number changes the color map. See *RGB*.
- color table** A workstation-dependent table in which the entries specify the values of the red, green and blue intensities defining a particular color, for use in mapping from a color index to a corresponding color. See *color map*, *RGB*.
- color value** The values of the RGB (red, green, and blue) components describing a color. See *RGB*.

Computer Graphics Interface

The Computer Graphics Interface (CGI) is a standard interface between device-dependent and device-independent code in a graphics environment. CGI makes all device drivers appear identical to the calling program. SCO CGI is based on CGI and all device drivers written for SCO CGI must conform to the CGI specification.

- control character** A nonprintable character, such as a carriage return, backspace, or bell, that controls an output device operation.
- control function** A function which allows you to exercise control over certain aspects of the system and the display device.
- coordinate scaling** A scaling function which transforms points from one *space* to another. In SCO CGI all point coordinates must be specified in Virtual Device Coordinates with values between -32768 and 32767. These coordinates are then scaled into values which are appropriate for your graphics device. See *Virtual Device Coordinate*, *Device Coordinate*.

- coordinate transformation** The act of transforming a particular location from one coordinate space to another. See *coordinate scaling*.
- cursor** A visual representation of the current addressable position.
- cursor keys** The keys you use to move the cursor on the screen.
- cursor text** Text which is meant to be displayed on a CRT screen. It is positioned by row and column in only one size, font, and orientation. Cursor text is displayed in cursor addressing mode only. Compare *alpha text*, *graphics text*.
- DC** See *Device Coordinate*.
- default** A function-dependent value assigned to a parameter by the program when no value is specified by the operator.
- Device Coordinate** A coordinate expressed in a coordinate system that is device-dependent.
- device coordinate unit** A unit of measure for the physical space represented by the display surface.
- device driver** Device-dependent software that communicates with a specific graphics device to draw graphics on the display surface based on Computer Graphics Interface functions.
- device handle** A number returned when the workstation is opened, which identifies a unique device. See *handle*.
- device independence** The ability to be used on more than one type of graphics display device.
- device space** The space defined by the addressable points of a graphics device.
- device units** The unit of resolution of the device; normally, a pixel. See *pixel*.

display device	A device (for example, a refresh display, storage tube display, or plotter) on which display images can be represented.
display surface	The part of a display device on which a visible image appears.
dot	The smallest visible point that can be displayed on the display surface. See <i>pixel</i> .
echo handle	The device handle used to display the graphics cursor on an echo device. See <i>handle</i> .
file	A collection of related information recorded on a diskette or a hard disk. The operating system determines how files are managed. Although a file may consist of many bytes of data residing in many disk tracks and sectors, a single file name identifies the entire file.
fill area	An output primitive consisting of a polygon (closed boundary) that may be hollow or may be filled with a uniform color, a pattern, or a hatch style.
fill interior style	The style in which any graphics primitive that honors fill area attributes is filled. It may be either Hollow , Solid , Hatch , or Pattern .
fill style index	The index indicating the hatch style or pattern style with which a fill area is filled.
font	A style and size of print for text output.
foreground color	The color in which graphics output primitives, fill area outlines, chart nomenclature, grid and tick marks, or chart and view area frames are displayed. Compare <i>background color</i> .
GDP	See <i>Generalized Drawing Primitive</i> .
Generalized Drawing Primitive	A display element (output primitive) used to address special geometrical workstation capabilities such as curve drawing.
graphics cursor	A cursor, often in the shape of a plus or a crosshair, that is used to enter graphics data.

- graphics device** A device, such as a refresh display, storage tube, graphics printer or plotter, on which graphics images can be displayed.
- graphics input device** A device, such as a keyboard, mouse, joystick, or tablet that transmits input data to the application.
- graphics input mode** An interactive mode in which a computer request causes the terminal to respond with graphics information. Status information or control characters may be part of the transmission.
- graphics mode** A mode on a graphics device which permits data to be interpreted as display-positioning information
- graphics output device**
A device, such as a printer, plotter, or display, that receives the transmitted graphics from the application
- graphics primitive** A graphics primitive is the basic graphics operation performed by SCO CGI (for example, drawing lines, markers, and text strings).
- graphics text** Text which is infinitely scalable, and which can be placed anywhere on the display surface. Its appearance may vary according to the output device's capabilities. Graphics text is displayed in graphics mode. Compare *alpha text*, *cursor text*.
- halfline** A horizontal line between the capline and the baseline within the character body, about which a horizontal string of characters would appear centrally placed in a vertical direction. All halflines in a font are in the same position in the character bodies. See *baseline*, *capline*, *character body*.
- handle** A number that uniquely specifies an object such as a bitmap, workstation, or device.
- hardcopy** A reproduction on paper of a terminal display.
- hard-copy device** A device for printing and plotting output on a piece of paper.

hardware font	The font built into the hardware device. See <i>font</i> .
hatch style	A method used to fill closed figures. A hatch style consists of one or more sets of lines filling the interior of a particular closed figure. The hatching lines include the boundary of a figure.
home	A starting or default position on the display surface.
host-independent	Capable of running on a number of operating systems.
hot spot	The reference point of a marker or cursor that corresponds to its specified position on the display surface. In the case of an arrow, an appropriate <i>hot spot</i> might be its tip. In the case of a cross, an appropriate <i>hot spot</i> might be its center.
input extent	The region of Virtual Device Coordinate space to which an input device's Device Coordinate space will be mapped. See <i>Device Coordinate</i> , <i>Virtual Device Coordinate</i> .
input function	A function that returns information from the operator. An input function returns the point location from the input device, the status of a valuator or a choice device, or allows input from the keyboard. It operates in sample and request modes.
inquiry function	A function that allows your program to determine the present state of the system. You can determine the current value of primitive attributes, device capabilities, device state, or error state.
integer	A whole number; a number with no fractional part.
integer pixel array	See <i>pixel array</i> .
joystick	An stick-shaped input device that can be moved in two dimensions to provide display-positioning information.
language binding	The specification of the exact calling syntax and data types for arguments to be used when calling

	functions from a specific programming language.
locator	See <i>locator input device</i> .
locator input device	A SCO CGI logical input device providing a position in Virtual Device Coordinates. For example, a mouse is a locator input device.
logical input device	An abstraction of one or more physical devices which deliver logical input values to the program.
logical input value	An abstraction of a physical value delivered by a physical input device such as a mouse button.
logical variable	A variable having only two possible values, <i>true</i> or <i>false</i> .
measure	A value associated with a logical input device which is determined by: <ul style="list-style-type: none"> • one or more physical input devices, and • a mapping from the values delivered by the physical device. <p>The logical input value delivered by the logical input device is the current value of the measure. See <i>logical input device</i>, <i>logical input value</i>.</p>
message	A string of characters used to communicate information to operators at metafile interpretation time.
metafile	A mechanism for retaining and transporting graphics data and control information. This information contains a device-independent description of one or more pictures.
mode	A state of a device affecting the interpretation of information which it receives, the format of information which it transmits, or its operation.

mouse	An input device that, when moved across a flat surface, moves the cursor correspondingly on the screen. The mouse usually has a button or buttons that can be pressed to signal locator termination.
nominal character	The prototype character around which a font is designed.
nominal line width	A workstation-dependent default width for a line. This value is multiplied by the line width scale factor and mapped to the workstation's nearest supported line width.
nominal marker size	A workstation-dependent default size for a marker. This value is multiplied by the marker size scale factor and mapped to the workstation's nearest supported marker size.
null-terminated string	A one-dimensional array or list of characters. The end of a string is indicated by the NULL character(ADE0).
opaque	A mode in which the background pixels of a graphics primitive are drawn using the current background color and writing mode. Compare <i>transparent</i> .
origin	The point of intersection of the <i>x</i> and <i>y</i> axes.
outline font	A character definition in which the edges of the character are defined by lines, arcs, and splines. Compare <i>bitmap font</i> . See <i>spline</i> .
output primitives	Functions that cause graphics to be drawn on a display surface. These functions describe polylines, polymarkers, text strings, pixel arrays, fill areas and generalized drawing primitives. The invocation of an output primitive function results in an output primitive, such as a sequence of markers or polylines. The appearance of output primitives is affected by the values of primitive attributes.
parameter	A value passed between a calling program and a routine.

pattern style	A method of filling closed figures with patterns. A pattern style consists of an array of variously colored or shaded points. The points may be the size of a pixel or larger. The pattern includes the boundary of a figure.
pel	See <i>pixel</i> .
physical bitmap	The bitmap, as displayed by the output device. See <i>bitmap</i> .
pixel	The smallest element of a display surface that can be independently addressed.
pixel array	An output primitive consisting of a rectangular grid of color indexes, each having a specified color. A color index can be either a byte or an integer value. These color indexes map one-to-one with frame buffer pixels.
polyline	A graphics primitive consisting of a series of connected line segments.
polymarker	A graphics primitive consisting of a set of marker symbols drawn at specified locations on a display surface.
raster	A field of closely spaced lines on the face of a video terminal that defines an image. The spacing between raster lines defines the resolution of a display.
raster graphics	Computer graphics in which the display image is composed of rasters. See <i>raster</i> .
raster font	See <i>bitmap font</i> , <i>bitmap</i> .
real number	A number that contains a fractional part expressed as a decimal; for example, 23.56.
request	A mode of input which causes the application program to pause and wait for user action. Compare <i>sample</i> .
RGB	An additive method for defining color in which tenths of percentages of the primaries red, green, and blue are combined to form other colors. Principal color combinations are given below.

	RED	GREEN	BLUE
Black	0	0	0
White	1000	1000	1000
Red	1000	0	0
Green	0	1000	0
Blue	0	0	1000
Yellow	1000	1000	0
Cyan	0	1000	1000
Magenta	1000	0	1000

- rotation Turning part or all of a display image about an axis.

- sample A mode of input operation in which an application polls the input device to see if input has occurred, and returns the current state of the input device. Compare *request*.

- scaling Enlarging or reducing part or all of a display image by multiplying the coordinates of a display image by a constant value; also, the transformation of points from one *space* to another.

- scan line The raster line drawn by the electron gun on a display screen in order to refresh the display. It may also be a row of a bitmap. See *raster*, *bitmap*.

- scan line conversion The process of converting a graphics primitive into the scan lines necessary to draw it on the display screen or into a bitmap. See *scan line*, *bitmap*.

- SCO CGI SCO CGI is a host- and device-independent graphics subsystem that serves as an environment for graphics applications as well as for application development.

- software font A font created through software. Text generated using a software font generally takes more time to print or display than text generated with a hardware font. This is because many calculations must be performed on each character before it can be drawn. See *font*.

spline	A mathematically defined curve.
string input device	A logical input device that provides a text string; for example, a keyboard.
stroke font	A character description in which the character is defined by lines.
text	An output primitive consisting of a character string.
text font and precision	An aspect of text having two components, font and precision, which together determine the shape of the characters being produced on a particular workstation.
text path	See <i>character up vector</i> .
transformation	The mapping of objects from one coordinate space to another; for example, from Virtual Device Coordinates to device coordinates. See <i>Virtual Device Coordinate, Device Coordinate</i> .
transparent	A mode in which the background color of the graphics primitives is ignored. Compare <i>opaque</i> .
trigger	A physical input device that an operator can use to indicate significant moments in time.
two's complement integers	A binary means of representing positive and negative numbers. The two's complement of a number is found by complementing every bit in the number (changing ones to zeros and vice versa) and adding one to the resulting value.
valuator input device	A logical input device that returns scalar values in a range (0 to 32767) that is proportional to the valuator position; for example, a control dial.
VDC	See <i>Virtual Device Coordinate</i> .
vector	A line.
vector graphics	See <i>calligraphic graphics</i> .
view surface	See <i>display surface</i> .

Viewport Specification Units

Coordinates that represent locations in a device-independent virtual space used by the SCO CGI. VSUs are associated with a viewport.

virtual bitmap A bitmap that may include or duplicate the image of a physical bitmap, in memory. It is not visible to the user. Compare *physical bitmap*. See *bitmap*.

virtual device An idealized graphics device that presents a uniform set of graphics capabilities to graphics software or systems through the Computer Graphics Interface.

Virtual Device Coordinate

A device coordinate space in which the full extent of the device axes are assigned values between -32768 and 32767. This convention provides improved device independence for a graphics system by allowing the viewing operations to be carried out without regard for device coordinate specifics. The VDC coordinates are transformed to specific device coordinates by SCO CGI.

Virtual Device Coordinate space

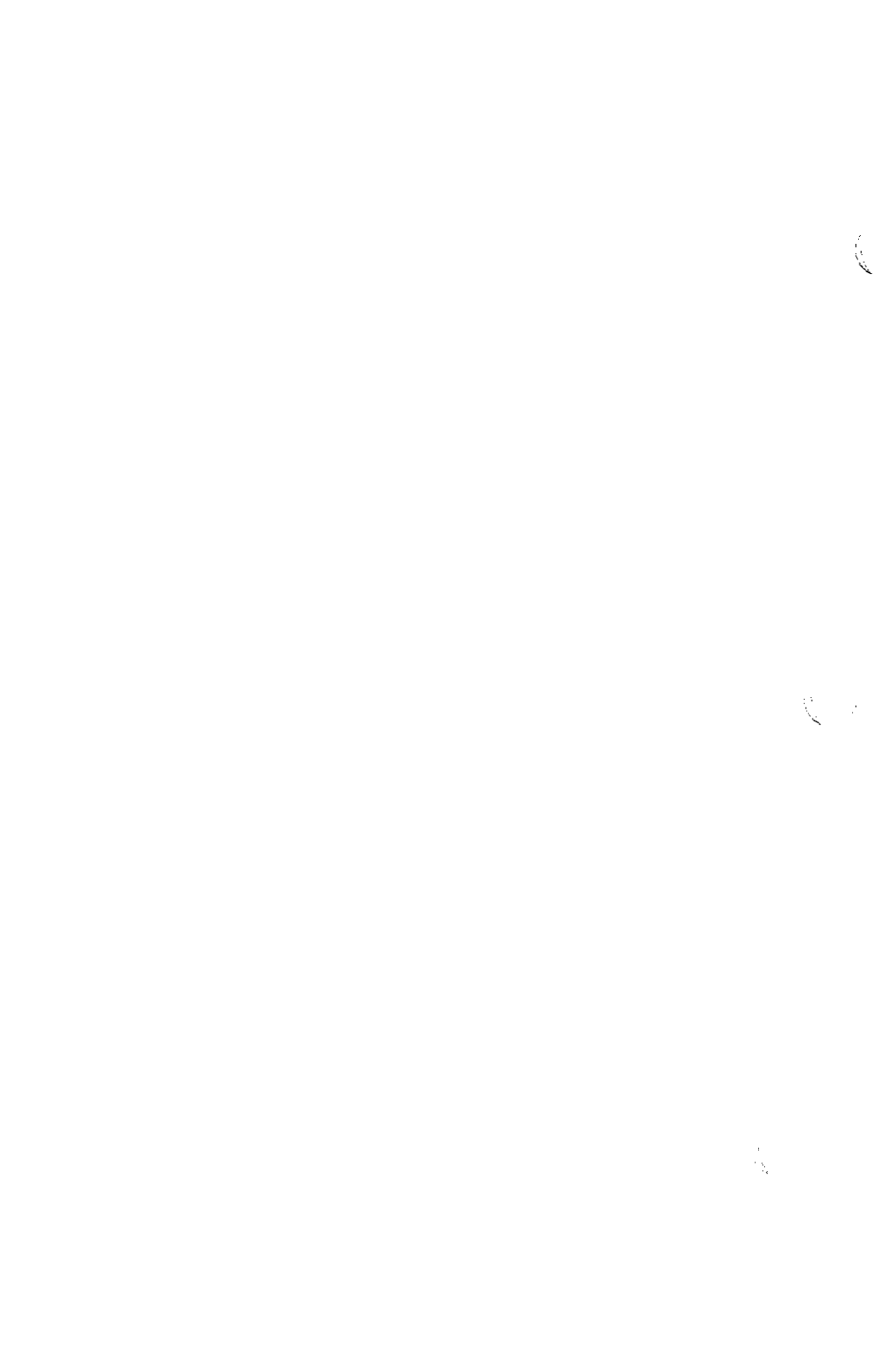
Virtual Device Coordinate space is a uniform virtual space by which a graphics application program passes graphics information to a device. SCO CGI transforms coordinates between VDC space and the device coordinates of a particular device.

workstation The logical interface through which the application program controls one or more input or output devices.

writing mode A logical operation defining the new value of a pixel in a bitmap, given its previous value, and a source pixel from a graphics primitive.

x/y axis system

A coordinate system composed of a horizontal *x* axis increasing positively toward the right and a vertical *y* axis increasing positively upwards. The axes are positioned at right angles and the point of intersection is the origin (0,0). The position of any point is defined by the displacement from the origin along first the *x* and then the *y* axis. See *Cartesian coordinate system*.



Index

A

Accessing font file information C-12
Alpha text 1-3, 2-8, 2-12
ANSI 1-5
Application data 3-7
Aspect ratio 2-4
Attribute functions 2-15, 3-3
Attributes 1-3

B

Background color 3-8, 3-199
Background mode 3-11
Bindings B-1
Bitblt hardware 3-101
Bitmap 1-3, 2-2, 2-3, 3-8, 3-15, 3-204
Bitmap extents 3-11
Bitmap functions 2-10, 3-3
Bitmap, monochrome 3-13
Bitmap, physical device 3-13
Bitmap, user-defined 3-9

C

Canonical rectangle 3-58, 3-244
CGI coordinate system 2-3
CGI error 3-131
CGI specific error codes A-1
CGM 1-3, 2-10, B-2
Character body 3-233
Character cell 3-187, 3-233
Character cell grid 3-32
Character cell position 3-43
Character descriptor C-8
Character positioning C-11
Choice input 2-16
Clear workstation 3-8
Clip rectangle 3-11, 3-54, 3-59, 3-96,
3-126, 3-150, 3-206, 3-246
Clipping 2-16
Close workstation 3-9
Color indices 3-52, 3-55, 3-95, 3-128,
3-206
Color table 3-207
Computer graphics metafile 1-3
Concatenation point 3-81
Control functions 2-9, 3-2
Coordinate transform mode 2-4

Coordinate transformation 1-2,
2-2, B-1
Coordinate transformation flag
2-4, 2-7
Copy bitmap 3-10
Copying screen to printer 3-37
Create bitmap 3-12
Create cursor 3-14
Creating graphics applications 1-4
Current cursor position 3-61
Cursor addressable text 2-8
Cursor addressing mode 2-8
Cursor, default 3-183
Cursor down 3-19
Cursor home 3-20
Cursor left 3-21
Cursor movement keys 2-16
Cursor, predefined 3-182
Cursor right 3-22
Cursor text 1-3, 2-8, 2-12
Cursor text maps 3-74
Cursor up 3-23
Cursor, user-defined 3-9

D

Dash pattern 3-254, 3-265
Default states 3-170
Defaults 3-114
Deferral mode 3-72, 3-212
Delete bitmap 3-24
Delete cursor 3-26
Descender 3-233
Device coordinates 1-2
Device drivers B-1
Device independent graphics 1-2
Device-driver management B-1
Device-specific data 2-8
Device-specific functions 2-9
Direct cursor address 3-27
Display graphics input cursor 3-29

E

Enter cursor addressing mode 3-31
Erase to end of line 3-33
Erase to end of screen 3-34
Error codes 2-17, A-1
Error handling B-2
Error messages A-1
Escape 3-35
Exit cursor addressing mode 3-36

Index

F

Font coordinate system 3-65,
3-234
Font emulation 3-187
Font file format C-1
Font file header C-3
Font header C-3
Font metrics C-6

G

Graphics emulation B-3
Graphics input 1-2, 1-3, 2-1
Graphics mode 2-8
Graphics model 2-1
Graphics output 1-2, 2-2
Graphics standards 1-5
Graphics standards and SCO CGI
1-5
Graphics text 1-3, 2-8, 2-12
GSS-DRIVERS 1-2

H

Hard copy 3-37
Hardware fonts 3-237
Hollow 3-219, 3-220
Home position 3-20
Hot spot 3-14, 3-70, 3-99

I

Initialization functions 2-9
Input functions 2-16, 3-4
Inquire addressable character cells
3-38
Inquire alpha text capabilities 3-39
Inquire alpha text cell location 3-42
Inquire alpha text font capability 3-44
Inquire alpha text position 3-47
Inquire alpha text string length 3-48
Inquire background mode 3-49
Inquire bitmap formats 3-50
Inquire byte pixel array 3-51
Inquire cell array 3-54
Inquire CGI error 3-56
Inquire clip rectangle 3-57
Inquire color representation 3-59
Inquire current cursor text address
3-61

Inquire current fill area attributes
3-62
Inquire current graphics text attributes
3-64
Inquire current line attributes 3-66
Inquire current marker attributes
3-68
Inquire cursor description 3-70
Inquire deferral mode 3-72
Inquire displayable bitmaps 3-73
Inquire drawing bitmap 3-75
Inquire fill area representation 3-77
Inquire graphics input cursor 3-79
Inquire graphics text extent 3-80
Inquire graphics text font character 3-83
Inquire graphics text font description
3-85
Inquire graphics text font metrics 3-88
Inquire graphics text representation
3-90
Inquire input extent 3-93
Inquire integer pixel array 3-94
Inquire line representation 3-96
Inquire marker representation 3-99
Inquire on cursor text attributes 3-210
Inquire on fill area representation 3-215
Inquire on graphics text representation
3-240
Inquire on line representation 3-252
Inquire on marker representation 3-261
Inquire optimum pattern size 3-101
Inquiry functions 2-17, 3-5
ISO 1-5

K

Kerned characters C-10

L

Language bindings 1-4
Load CGI 3-103
Locator input 2-16

M

Master function list 3-1
Message 3-106
Metafile 1-3, 2-10, 3-106
Metafile B-2
Metafile generator 3-7
Microjustification 3-48

O

Opaque 3-202, 3-248
 Open workstation 3-108
 Output alpha text 3-116
 Output arc 3-118
 Output bar 3-122
 Output byte pixel array 3-124
 Output cell array 3-127
 Output CGI error 3-131
 Output circle 3-132
 Output cursor addressable text
 3-134
 Output ellipse 3-136
 Output elliptical arc 3-138
 Output elliptical pie slice 3-141
 Output filled area 3-143
 Output functions 2-10, 3-3
 Output graphics text 3-146
 Output integer pixel array 3-148
 Output pie slice 3-150
 Output polyline 3-152
 Output polymarker 3-155

P

Page 3-74
 Pass through mode 3-192
 Pattern size 3-101
 Pel2-3
 Picture element 2-3
 Pixel 1-4, 2-3
 Pixel array 2-12
 Predefined cursors 3-71
 Primitives 1-2, 2-15
 Program construction 2-6
 Prompt 3-8
 Prompting flag 2-8
 Proportional fonts 3-48
 Proportional spacing 3-233
 Proportional spacing flag 3-187
 Pseudocode 3-1

R

Raster 2-2
 Raster address mode 2-5
 Raster font file architecture C-1
 Raster technology 2-2
 Read cursor keys 3-158
 Reference point 3-14, 3-70
 Remove CGI 3-161
 Remove graphics input cursor 3-162

Request choice 3-163
 Request locator 3-164
 Request locator 3-29
 Request mode 2-16
 Request string 3-167
 Request valuator 3-169
 Reset to defaults 3-170
 Reverse video off 3-172
 Reverse video on 3-173

S

Sample choice 3-174
 Sample locator 3-175
 Sample mode 2-16
 Sample string 3-178
 Sample valuator 3-180
 SCO product environment 1-4
 SCO CGI 1-2
 Select drawing bitmap 3-181
 Select graphics input cursor 3-182
 Set alpha text color index 3-184
 Set alpha text font and size 3-186
 Set alpha text line spacing 3-189
 Set alpha text overstrike mode
 3-191
 Set alpha text pass through mode
 3-192
 Set alpha text position 3-193
 Set alpha text quality 3-194
 Set alpha text sub/superscript mode
 3-196
 Set alpha text underline mode 3-198
 Set background color index 3-199
 Set background mode 3-201
 Set clip rectangle 3-203
 Set color representation 3-205
 Set color table 3-207
 Set cursor text attributes 3-208
 Set cursor text color index 3-211
 Set deferral mode 3-212
 Set fill area representation 3-214
 Set fill color index 3-216
 Set fill interior style 3-218
 Set fill style index 3-220
 Set graphics text alignment 3-227
 Set graphics text character height
 3-232
 Set graphics text color index 3-235
 Set graphics text font 3-237
 Set graphics text representation
 3-239
 Set graphics text string baseline
 rotation 3-241
 Set input extent 3-243
 Set line color index 3-245
 Set line cross section 3-247

Index

Set line edit characters 3-249
Set line representation 3-250
Set line type 3-253
Set line width 3-255
Set marker color index 3-257
Set marker height 3-259
Set marker representation 3-260
Set marker type 3-262
Set pen speed 3-264
Set user line type 3-265
Set writing mode 3-267
Solid 3-219, 3-220
String input 2-16
System dependent error messages
 A-1

T

Termination functions 2-9
Text alignment point 3-147
Text character height 3-233
Text concatenation 3-81
Text extent rectangle 3-228
Text modes 1-3
Text size 2-8
Transform mode 0 (full screen mode)
 2-5
Transform mode 1 (preserve aspect
 ratio mode) 2-5
Transform mode 2 (device units mode)
 2-5
Transform mode 3 (short axis mode)
 2-5
Transformation modes 3-58
Transparent 3-202
Trigger 2-16

U

Update Workstation 3-270
User-specified line type 3-96, 3-252
User-specified marker type 3-99, 3-260
Using SCO CGI 1-2
Using the open workstation function 2-7
Using the SCO CGI programmer's
 guide 2-17

V

Valuator input 2-16
VDC 1-2, 2-2, 2-3
VDC space 2-3
Video buffer 3-13
Virtual device coordinates 1-2, 2-2,
 2-3

W

Workstation 2-1, 3-114
Workstation control functions 2-10
Writing mode 3-11, 3-125, 3-149

Replace this Page
with Tab Marked:

**Device Driver
Supplement**



SCO XENIX®

Development System

Device Driver Supplement

1

2

3

Information in this document is subject to change without notice and does not represent a commitment on the part of The Santa Cruz Operation, Inc. nor Microsoft Corporation. The software described in this document is furnished under a license agreement or non-disclosure agreement. The software may be used or copied only in accordance with the terms of the agreement. It is against the law to copy this software on magnetic tape, disk, or any other medium for any purpose other than the purchaser's personal use.

Portions © 1986, 1987 Graphic Software Systems, Inc.

All rights reserved.

Portions © 1987 The Santa Cruz Operation, Inc.

All rights reserved.

ALL USE, DUPLICATION, OR DISCLOSURE WHATSOEVER BY THE GOVERNMENT SHALL BE EXPRESSLY SUBJECT TO RESTRICTIONS AS SET FORTH IN SUBDIVISION (b) (3) (ii) FOR RESTRICTED RIGHTS IN COMPUTER SOFTWARE AND SUBDIVISION (b) (2) FOR LIMITED RIGHTS IN TECHNICAL DATA, BOTH AS SET FORTH IN FAR 52.227-7013.

This document was typeset with an IMAGEN® 8/300 Laser Printer.

Microsoft and XENIX are registered trademarks of Microsoft Corporation.

IMAGEN is a registered trademark of IMAGEN Corporation.

GSS and the GSS logo are registered trademarks, and GSS*CGI and GSS*GRAFSTATION are trademarks of Graphic Software Systems, Incorporated.

IBM is a registered trademark of International Business Machines, Incorporated.

SCO Document Number: XG-6-12-87-1.0

Processed: Wed Jun 10 14:18:19 PDT 1987

Contents

1 Overview

- 1.1 About This Manual 1-1
- 1.2 SCO CGI Device Drivers 1-1
- 1.3 Device Driver Management 1-2

2 Device-Specific Features

- 2.1 A Note About Device Features 2-1
- 2.2 Device-Specific Features 2-1

3 Display Devices

- 3.1 Features Common To Display Devices 3-1
- 3.2 IBM Color Graphics Adapter - High Resolution Monochrome 3-9
- 3.3 IBM Color Graphics Adapter - Medium Resolution Color 3-12
- 3.4 IBM Enhanced Graphics Adapter 3-16

4 Hard Copy Devices

- 4.1 Features Common To Hard Copy Devices 4-1
- 4.2 Apple LaserWriter 4-4
- 4.3 Epson 80 Series Printers 4-7
- 4.4 Epson 100 Series Printers 4-10
- 4.5 Hewlett-Packard Plotters 4-14
- 4.6 Hewlett-Packard Laserjet Printer 4-16
- 4.7 Hewlett-Packard Thinkjet Printer 4-20

5 Graphics Input Devices

- 5.1 Features Common To Graphics Input Devices 5-1

6 Other Types of Devices

- 6.1 About These Devices 6-1
- 6.2 Computer Graphics Metafile (CGM) 6-2
- 6.3 GSS*GRAFSTATION 6-4

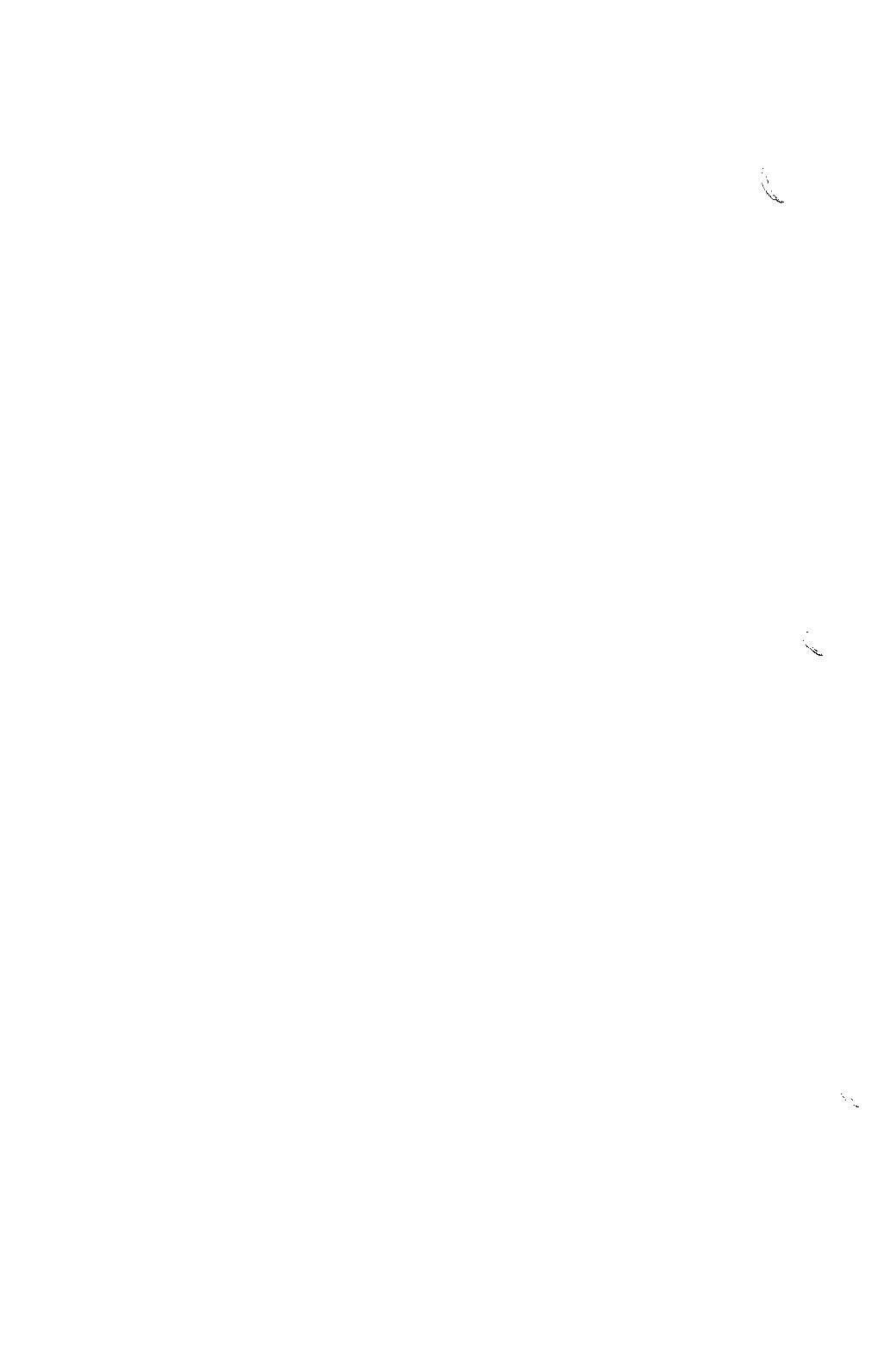
G Glossary



Chapter 1

Overview

- 1.1 About This Manual 1-1
- 1.2 SCO CGI Device Drivers 1-1
- 1.3 Device Driver Management 1-2



1.1 About This Manual

The *SCO CGI Device Driver Supplement* is intended for users and developers of graphics application programs. SCO CGI allows graphics application programs to operate with many kinds of peripheral devices such as plotters, dot matrix printers, and displays. Each peripheral device has a unique set of capabilities and specifications. The purpose of a device driver is to support these capabilities. This supplement provides general and device-specific information for the peripheral devices supported by SCO CGI.

SCO graphics-based applications programs access many of the SCO CGI functions. Refer to the *SCO CGI Programmer's Guide* for details on using SCO CGI in your own programs.

Chapter two describes the organization of the information in the following chapters.

Chapters three through six present the information, divided into four classes of devices:

- Displays,
- HardCopy,
- Graphics Input,
- Other.

In addition to this manual, the following SCO CGI publications are also available:

- The *SCO CGI Programmer's Guide* contains detailed generic descriptions of each of the SCO CGI functions, and provides installation instructions.
- Programming language-specific binding reference booklets accompany the *SCO CGI Programmer's Guide*, providing the language-specific syntax for each SCO CGI function.

1.2 SCO CGI Device Drivers

To enable the programmer to use a variety of devices, software must translate between the application program and the peripheral devices. Each device requires a different translation. SCO CGI therefore provides a translator, or driver, for each device.

SCO CGI Device Driver Supplement

SCO CGI is a package of drivers that implements the Computer Graphics Interface on your system for graphics devices. Figure 1-1 shows the role of device drivers within the SCO CGI graphics architecture.

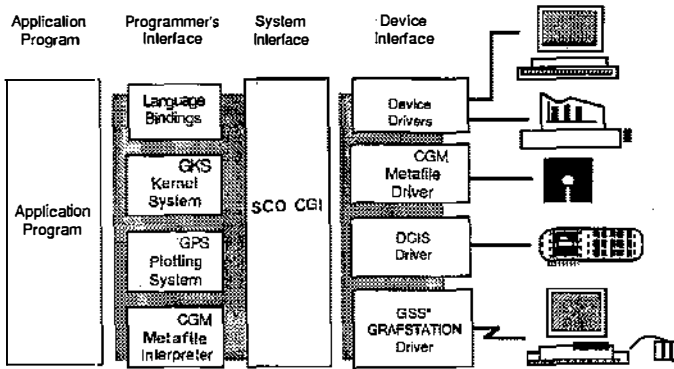


Figure 1-1 *Device driver's role within SCO CGI*

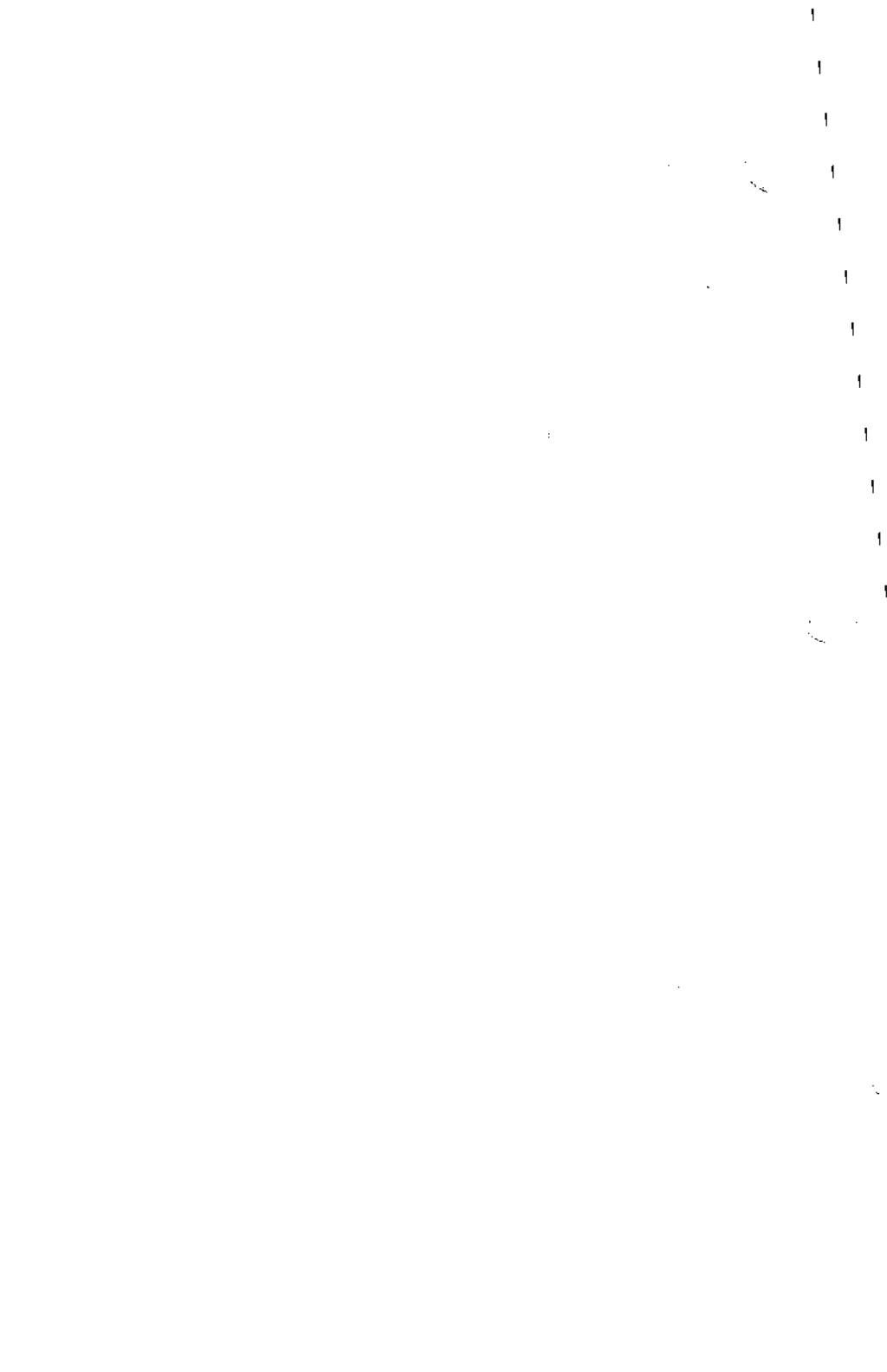
1.3 Device Driver Management

Your system can have many graphics input and output peripherals attached, each requiring a specific device driver to make it work with the system. SCO CGI receives requests for graphics peripherals from an application program and loads the device driver into memory.

Chapter 2

Device- Specific Features

- 2.1 A Note About Device Features 2-1
- 2.2 Device-Specific Features 2-1
 - 2.2.1 Filename 2-1
 - 2.2.2 Device Logical Name 2-2
 - 2.2.3 Environmental Settings 2-2
 - 2.2.4 Features Supported 2-2



2.1 A Note About Device Features

Devices supported by SCO CGI are grouped into four classes and described in the following chapters:

- Display devices, Chapter 3,
- Hard copy devices, Chapter 4,
- Graphics input devices, Chapter 5,
- Other devices, Chapter 6.

The devices in each of the four groups have common features. For example, entering strings or drawing lines are tasks common to the entire group of display devices. There are also features specific to each device in the group. Examples of such features specific to display devices are the management of cursor addressable text or the color palette support. This specific information is included in the following chapters for each of the following devices supported by SCO CGI :

Table 2-1. Devices supported by SCO CGI

DEVICE NAME	FILE NAME
Hard Copy Devices	
Apple LaserWriter	laserwriter
Epson 80 Series Printer	epson80
Epson 100 Series Printer	epson100
Hewlett-Packard Plotters	hpplot
Hewlett-Packard Laserjet Printer	laserjet
Hewlett-Packard Thinkjet Printer	thinkjet
Display Devices	
IBM CGA High Resolution Monochrome	cgabw
IBM CGA Medium Resolution Color	cgacö
IBM EGA	ega
Other Devices	
Computer Graphics Metafile	ddmeta
GSS*GRAFSTATION Driver	gstdd

2.2 Device-Specific Features

Each device is described in a device driver information sheet. The following information is included in each device driver information sheet.

2.2.1 Filename

The name of the device driver file.

2.2.2 Device Logical Name

Each device is a member of one of the following classes: *Displays*, *Hard Copy*, *Graphics Input*, or *Other*. The names **DISPLAY**, **PRINTER**, **PLOTTER**, **MOUSE**, **TABLET**, or similar appropriate name can be used to identify the device.

2.2.3 Environmental Settings

Most devices have special states in which they can operate. A printer, for example, can support narrow or wide paper by setting the environmental variable **PAPER** to either narrow or wide.

2.2.4 Features Supported

This section describes the SCO CGI features supported on the device. Device-specific features include:

- **Polylines:** line styles available and their indices.
- **Graphics Cursors:** graphics input cursors available and their indices.
- **Polymarkers:** marker styles available and their indices.
- **Fill Areas:** styles available for filling in primitives, and their corresponding indices.
- **Color:** color capabilities of the device, default colors, color selection, color indices and in the case of plotters, color index to pen correspondence.
- **Set Input Extent:** the input extent rectangle which controls the range of coordinates returned by the **REQUEST LOCATOR** and **SAMPLE LOCATOR** functions.
- **Request Locator:** locator capabilities of the device, such as graphics input cursor, positioning, and manipulation.
- **Sample Locator:** return of locator information.
- **Request Choice:** choice input method and mapping of choice values to function keys.

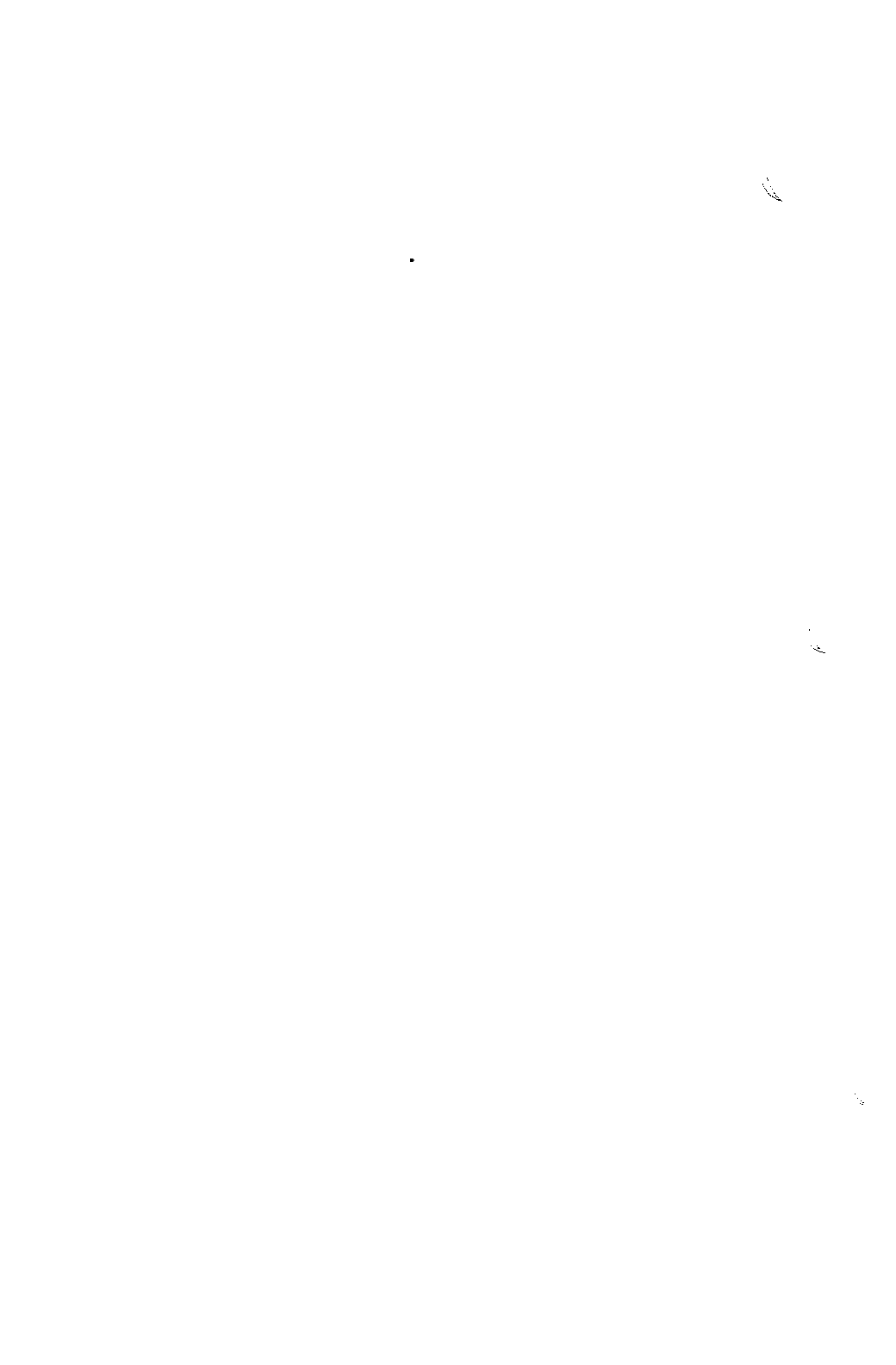
- **Sample Choice:** return of choice values and mapping to function keys.
- **RequestString:** method of returning strings.
- **Sample String:** return of current string.
- **Cursor Addressable Text:** color and other attributes.
- **Alpha Text:** alpha text capabilities of the device, such as fonts, sizes, color and other attributes.
- **Graphics Text:** graphics text capabilities of the device, such as character size and rotation.



Chapter 3

Display Devices








- 3.1 Features Common To Display Devices 3-1
 - 3.1.1 Polylines 3-1
 - 3.1.2 GraphicsCursors 3-1
 - 3.1.3 Polymarkers 3-1
 - 3.1.4 FillAreas 3-2
 - 3.1.5 Color 3-2
 - 3.1.6 RequestLocator 3-2
 - 3.1.7 SampleLocator 3-2
 - 3.1.8 RequestChoice 3-2
 - 3.1.9 SampleChoice 3-3
 - 3.1.10 RequestString 3-3
 - 3.1.11 SampleString 3-3
 - 3.1.12 Cursor-AddressableText 3-3
 - 3.1.13 AlphaText 3-3
 - 3.1.14 GraphicsText 3-4
- 3.2 IBMColor Graphics Adapter - High Resolution Monochrome 3-9
- 3.3 IBMColor Graphics Adapter - Medium Resolution Color 3-12
- 3.4 IBM Enhanced Graphics Adapter 3-16



3.1 Features Common To Display Devices

3.1.1 Polyline

Seven line styles are predefined:

1		Solid
2		Long Dash
3		Dotted
4		Dash Dotted
5		Medium Dashed
6		Dash with two dots
7		Short Dash

Line styles can also be user-defined. Refer to the Set User Polyline Line Type function in the *SCO CGI Programmer's Guide* for this information.

3.1.2 Graphics Cursors



Display devices support six predefined graphics-input cursors:

0. Crosshair (default)
1. Arrow
2. Checkmark
3. Pointing hand
4. Palm of hand
5. Hourglass

In addition, user-specified cursors are available. Refer to the Create Cursor function in the *SCO CGI Programmer's Guide* for more information.

3.1.3 Polymarkers

The following six graphics markers are available on all displays:

1. .
2. +
3. *
4. 
5. X
6. 

SCO CGI Device Driver Supplement

In addition user-specified markers are available. Refer to the Set Poly-marker Representation function in the *SCO CGI Programmer's Guide* for more information.

3.1.4 Fill Areas

The following fill areas are supported:

- Hollow (outlined).
- Solid.
- Six hatch styles. (See Figure 3-1.)
- Thirty-three pattern styles. (See Figure 3-2.)
- User-defined patterns.

Refer to the Set Fill Area Representation routine in the *SCO CGI Programmer's Guide* for more information about user-specified patterns.

3.1.5 Color

Refer to the specific device-driver information sheet.

3.1.6 Request Locator

When the locator is invoked, a graphics-input cursor appears on the screen at the initial locator position. Refer to the device-driver information sheet for information about cursor motion.

3.1.7 Sample Locator

When locator is invoked, the current position is returned without operator action.

3.1.8 Request Choice

The available function keys are used to enter choice input. Refer to the device-driver information sheet for mapping of choice values to the function keys.

3.1.9 Sample Choice

This returns any pending choice input. Refer to the device-driver information sheet for mapping of choice values to the functions keys.

3.1.10 RequestString

The keyboard is used to enter strings. The string is terminated by pressing the ENTER, RETURN or other corresponding key. The key to use is specified in the device-driver information sheet. The termination character is not returned with the string.

3.1.11 Sample String

All available characters are returned.

3.1.12 Cursor-Addressable Text

To perform cursor-control functions, the device must be in Cursor-Addressing Mode. To display graphics primitives, the device must be removed from Cursor-Addressing Mode.

Refer to the device-driver information sheet to determine the specific attributes available.

3.1.13 Alpha Text

You can position alpha text anywhere on the output page. The following text capabilities are supported:

- Font.
- Size.
- Superscript and subscript.
- Underline.
- Line spacing.

3.1.14 Graphics Text

Hardware text is available in five predefined heights. They are available in 0, 90, 180, and 270-degree rotations.

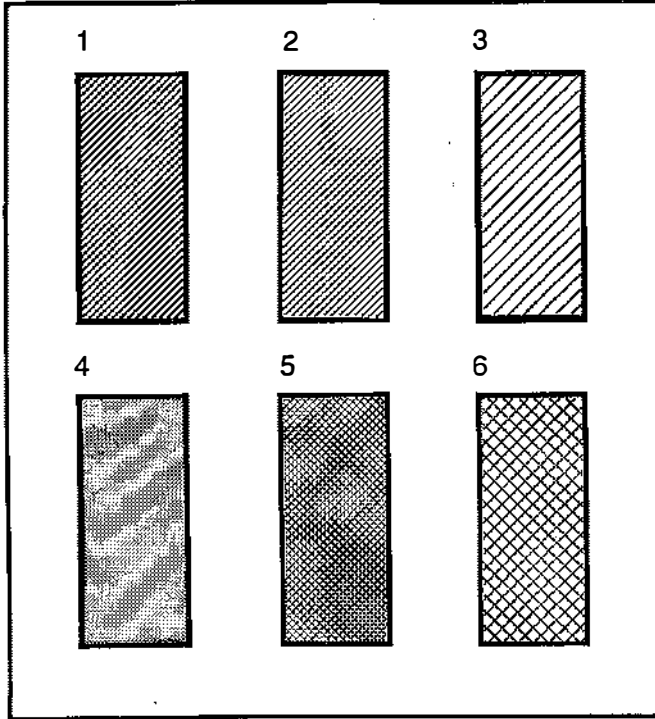


Figure 3-1 *Hatch Styles*

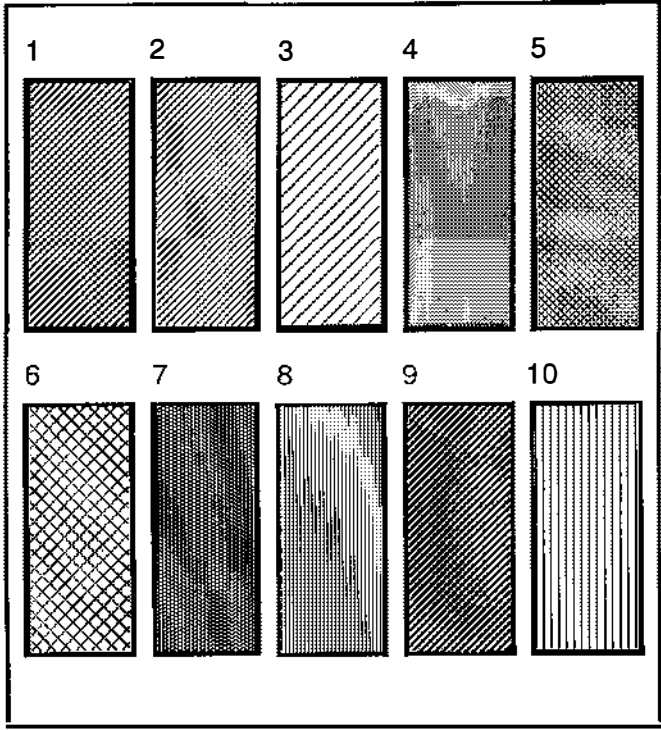


Figure 3-2 *Pattern Styles*

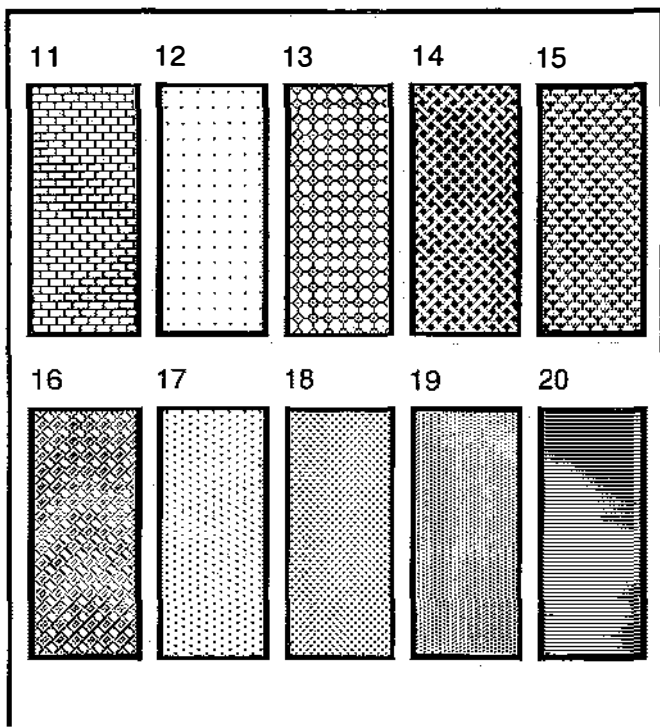


Figure 3-2 *Pattern Styles (continued)*

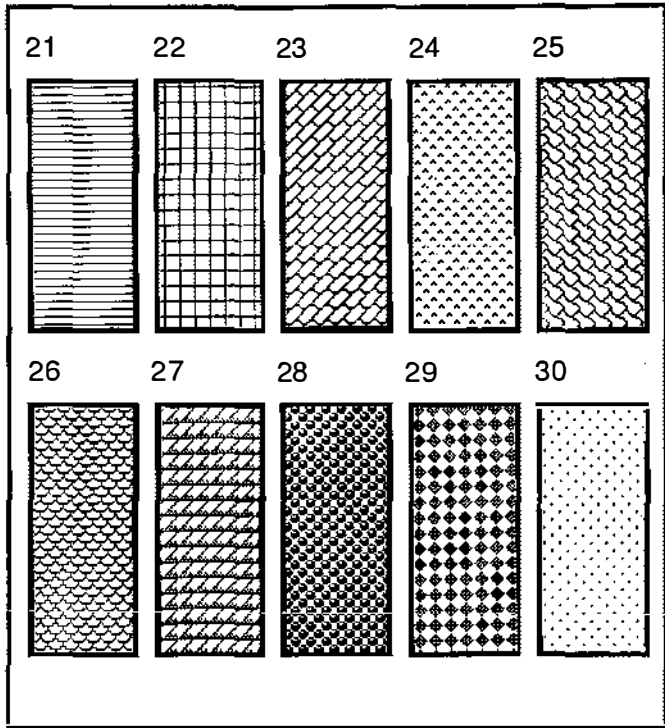


Figure 3-2 *Pattern Styles (continued)*

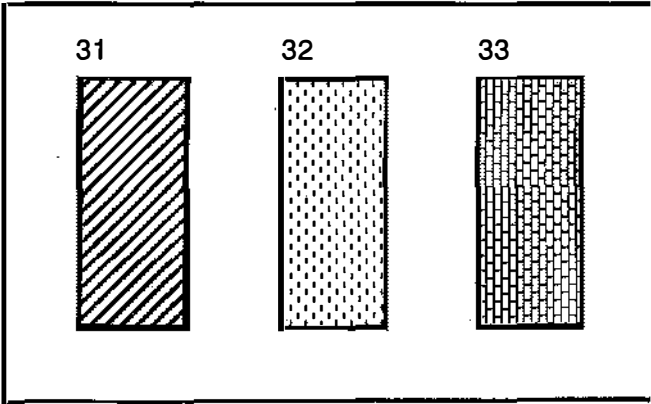


Figure 3-2 *Pattern Styles (continued)*

3.2 IBM Color Graphics Adapter- High Resolution Monochrome

Filename

cgabw

Device Logical Name

Display

Communications

not applicable

Features Supported

Color

In monochrome mode, the IBM supports two colors: Index 1 is displayed in white and index 0 is displayed in the background color. These colors cannot be redefined.

Request Locator

When locator is invoked, a tracking cross appears on the screen at the initial locator position. The cross can be moved by pressing one of the following eight keys on the numeric keypad: 1 (down & left), 2 (down), 3 (down & right), 4 (left), 6 (right), 7 (up & left), 8 (up), and 9 (up & right). The Numeric Lock function must be off for the cross to be moved. Initially, the cross moves in large increments. Pressing the 5 (INS) key toggles the distance between large movements and small movements. When the cross is at the desired location, the point can be selected by pressing any alpha key on the keyboard. This causes the coordinates of the point to be transmitted back to the user program. If desired, the device will perform an inking function. When the locator is terminated, a line from the initial position to the desired position is drawn, honoring the current line attributes such as color and line style.

The device also performs rubberbanding if desired. There are two types of rubberbanding supported, lines and boxes. If rubberbanding lines are desired, then a line will be drawn from the initial locator position to the current position of the graphics cursor. The line changes dynamically as the cursor is moved. When the locator is terminated, the line is removed. If rubberband rectangle is specified, a rectangle is displayed with one corner at the initial locator position and the opposite corner at the current position of the graphics cursor. The rectangle changes dynamically as the cursor is moved. When the locator is terminated, the rectangle is removed from the display.

Hard Copy

Hard Copy is not supported under XENIX.

Request Choice

The function keys F1 to F10 are used to enter choice input.

Request String

The keyboard is used to enter strings. A string is terminated by the ENTER key.

Cursor-Addressable Text

Cursor-addressable text is supported. The device must be in Cursor-Addressing Mode before it can perform any cursor control functions. The following attributes are supported:

- Reverse Video
- Blink
- Bold Intensity
- Color = Expanded palette

Color Index	Standard	Bold
0	Black	Dark Grey
1	Lt. Grey	White
2	Red	Lt. Red
3	Green	Lt. Green
4	Blue	Lt. Blue
5	Brown	Yellow
6	Cyan	Lt. Cyan
7	Magenta	Lt. Magenta

To display graphics primitives, the device must be removed from Cursor-Addressing Mode.

Graphics Markers

Markers have five sizes as listed below in NDC coordinates:

Preserve Aspect Ratio Mode

- 1 797
- 2 1480
- 3 2162
- 4 2845
- 5 3528

Non-Preserve Aspect Ratio Mode

- 1 1147
- 2 2130
- 3 3113
- 4 4096
- 5 5080

Additional Information

Upon termination, the drivers return the display to the initial operating mode.

3.3 IBM Color Graphics Adapter- Medium Resolution Color

Filename

cgaco

Device Logical Name

Display

Communications

not applicable

Features Supported

Color

The IBM color mode supports four colors in graphics mode. They are defined at Open Workstation time as follows:

- Index 0 Black(background)
- Index 1 White
- Index 2 Magenta
- Index 3 Cyan

There are four different color palettes available in the IBM medium-resolution device driver. Based on the user's desired color representation of indices 1, 2, and 3, a palette selection is made.

Index	Palette One	Palette Two	Palette Three	Palette Four
1	Brown	Yellow	Lt. Grey	White
2	Red	Lt. Red	Magenta	Lt. Magenta
3	Green	Lt. Green	Cyan	Lt. Cyan

Color index 0 can be mapped to any of the 16 available colors.

The actual RGB values for the possible colors follows:

R	G	B	COLOR	DEFAULT-INDEX
0	0	0	Black	0
0	0	600	Blue	
0	600	0	Green	
0	600	600	Cyan	
600	0	0	Red	
600	0	600	Magenta	
600	600	0	Brown	
600	600	600	Light Grey	
400	400	400	Dark Grey	
400	400	1000	Light Blue	
400	1000	400	Light Green	
400	1000	1000	Light Cyan	3
1000	400	400	Light Red	
1000	400	1000	Light Magenta	2
1000	1000	400	Yellow	
1000	1000	1000	White	1

RequestLocator

When locator is invoked, a tracking cross appears on the screen at the initial locator position. The cross can be moved by pressing one of the following eight keys on the numeric keypad: 1 (down & left), 2 (down), 3 (down & right), 4 (left), 6 (right), 7 (up & left), 8 (up), and 9 (up & right). The Numeric Lock function must be off for the cross to be moved. Initially, the cross moves in large increments. Pressing the 5 (INS) key toggles the distance between large movements and small movements. When the cross is at the desired location, the point can be selected by pressing any alpha key on the keyboard. This causes the coordinates of the point to be transmitted back to the user program. If desired, the device will perform an inking function. When the locator is terminated, a line from the initial position to the desired position is drawn honoring the current line attributes such as color and line style.

The device also performs rubberbanding if desired. There are two types of rubberbanding supported, lines and boxes. If rubberbanding lines are desired, then a line will be drawn from the initial locator position to the current position of the graphics cursor. The line changes dynamically as the cursor is moved. When the locator is terminated, the line is removed. If rubberband rectangle is specified, a rectangle is displayed with one corner at the initial locator position and the opposite corner at the current position of the graphics cursor. The rectangle changes dynamically as the cursor is moved. When the locator is terminated, the rectangle is removed from the display.

HardCopy

Hard Copy is not supported under XENIX.

RequestChoice

The function keys **F1** to **F10** are used to enter choice input.

RequestString

The keyboard is used to enter strings. The string is terminated by the **ENTER** key.

Cursor-Addressable Text

Cursor-addressable text is supported. The device must be in Cursor-Addressing Mode before it can perform any cursor-control functions. The following attributes are supported:

- Reverse Video
- Blink
- Bold Intensity
- Color = Expanded palette

Color Index	Standard	Bold
0	Black	Dark Grey
1	Light Grey	White
2	Red	Light Red
3	Green	Light Green
4	Blue	Light Blue
5	Brown	Yellow
6	Cyan	Light Cyan
7	Magenta	Light Magenta

To display graphics primitives, the device must be removed from Cursor-Addressing Mode.

Graphics Markers

Markers have five sizes as listed below in NDC coordinates:

Preserve Aspect Ratio Mode

1	797
2	1480
3	2162
4	2845
5	3528

Non-Preserve Aspect Ratio Mode

1	1147
2	2130
3	3113
4	4096
5	5080

Additional Information

Upon termination, the drivers return the display to the initial operatingmode.

3.4 IBM Enhanced Graphics Adapter

Filename

ega

Device Logical Name

Display

Default Resolution and Aspect Ratio

The Horizontal and Vertical dpi are used for selection of raster fonts.

Resolution	Aspect Ratio	Hor. dpi	Vert. dpi
640x350	1.27:1	75	59
640x200	2.36:1	98	42
320x200	1.18:1	49	42

Communications

not applicable

Features Supported

Environmental Settings

The user can specify the EGA mode by using the `sh` shell `set` or `cs` shell `setenv` command. For example, if 320x200 16-color mode is wanted, from the `sh` shell, type:

```
set EGA=MR3
export EGA
```

From the `cs` shell, type:

```
setenv EGA MR3
```

Color

Sixteen Color (MR3 & HR3)

This device supports the following sixteen colors:

Index 0	Black
Index 1	White
Index 2	Red
Index 3	Green
Index 4	Blue
Index 5	Yellow
Index 6	Cyan
Index 7	Magenta
Index 8-15	White

Color index can be mapped to any of the 16 available colors. The actual RGB values for the possible colors follow:

R	G	B	COLOR	DEFAULT-INDEX
0	0	0	Black	0
0	0	600	Blue	4
0	600	0	Green	3
0	600	600	Cyan	6
600	0	0	Red	2
600	0	600	Magenta	7
600	600	0	Brown	
600	600	600	Light Grey	
400	400	400	Dark Grey	
400	400	1000	Light Blue	
400	1000	400	Light Green	
400	1000	1000	Light Cyan	
1000	400	400	Light Red	
1000	400	1000	Light Magenta	
1000	1000	400	Yellow	5
1000	1000	1000	White	1(8-15)

These colors can be redefined. Each of the Red, Green, and Blue components can have 0, 400, 600, or 1000 values. This allows 64 colors. The SET COLOR REPRESENTATION function will set only an individual color index and not establish a palette. The entire palette may be set with the SET COLOR TABLE function.

Four or Sixteen color (HR4)

The IBM Enhanced Graphics Adapter with Enhanced Color Display device driver supports the four or sixteen colors depending on the amount of memory available on the card.

Four-color mode (64K)

Index 0	Black
Index 1	White
Index 2	Red
Index 3	Green

Sixteen color mode
(128K or 256K)

Index 0	Black
Index 1	White
Index 2	Red
Index 3	Green
Index 4	Blue
Index 5	Yellow
Index 6	Cyan
Index 7	Magenta
Index 8-15	White

These colors can be redefined. Each of the Red, Green, and Blue components can have 0, 400, 600, or 1000 values. This allows 64 colors. The SET COLOR REPRESENTATION function will set only an individual color index and not establish a palette. The entire palette may be set with the SET COLOR TABLE function.

Monochrome (MONO)

The IBM Enhanced Graphics Adapter with a Monochrome Monitor device driver supports the following four attributes as *colors*:

Index 0	Background
Index 1	Video
Index 2	Video Blink
Index 3	Bold Video

These color attributes cannot be redefined.

Request Locator

When locator is invoked, a graphics input cursor appears on the screen at the initial locator position. The cursor can be moved by pressing one of the following keys on the numeric keypad: **1** (down & left), **2** (down), **3** (down & right), **4** (left), **6** (right), **7** (up & left), **8** (up), and **9** (up & right). The Numeric Lock function must be off for the cursor to be moved. Initially, the cursor moves in large increments. Pressing the **5** (INS) key toggles the distance between large movements and small movements. When the cross is at the desired location, the point can be selected by pressing any alpha key on the keyboard.

Request Choice

This driver supports the following function key values:

Function	Value
F1-F10	1-10
Shift F1-F10	11-20
Ctrl F1-F10	21-30
Alt F1-F10	31-40

Cursor-Addressable Text

In addition to the common features, this device supports the following attributes:

Color= Expanded palette

Color Index	Standard	Bold
0	Black	Dark Grey
1	Lt. Grey	White
2	Red	Lt. Red
3	Green	Lt. Green
4	Blue	Lt. Blue
5	Brown	Yellow
6	Cyan	Lt. Cyan
7	Magenta	Lt. Magenta



Chapter 4

Hard Copy Devices

- 4.1 Features Common To Hard Copy Devices 4-1
 - 4.1.1 X/Y Orientation of Output 4-1
 - 4.1.2 Polylines 4-1
 - 4.1.3 Polymarkers 4-1
 - 4.1.4 Fill Areas 4-2
 - 4.1.5 Color 4-2
 - 4.1.6 Alpha Text 4-2
 - 4.1.7 Graphics Text 4-3
 - 4.1.8 Paper Handling 4-3
- 4.2 Apple LaserWriter 4-4
- 4.3 Epson80Series Printers 4-7
- 4.4 Epson100Series Printers 4-10
- 4.5 Hewlett-Packard Plotters 4-14
- 4.6 Hewlett-Packard Laserjet Printer 4-16
- 4.7 Hewlett-Packard Thinkjet Printer 4-20










4.1 Features Common To Hard Copy Devices

4.1.1 X/Y Orientation of Output

Two possible orientations for the output from hard copy devices are defined by the environmental settings: *portrait* (the *y* measurement is greater than the *x* measurement) or *landscape* (the *x* measurement is greater than the *y* measurement). For printers, if the environmental variable, PAPER, is set to NARROW, then the default orientation is portrait. If PAPER is set to WIDE, then the default orientation is landscape.

4.1.2 Polylines



Seven line styles are predefined for printers:

1		Solid
2		LongDash
3		Dotted
4		Dash Dotted
5		Medium Dash
6		Dash with two dots
7		Short Dash

Only the first six line styles above are predefined for plotters. Line styles can also be user-defined for printers. Refer to the Set User Polyline Line Type routine in the *SCO CGI Programmer's Guide* for this information.

4.1.3 Polymarkers

The following six graphics markers are available on all plotters and printers:

1. .
2. +
3. *
4. 
5. X
6. 

4.1.4 Fill Areas

The following fill areas are supported:

- Hollow (outlined).
- Solid.
- Six hatch styles (see Figure 3-1).
- Thirty-three pattern styles for printers (see Figure 3-2), six for plotters (these are the same as the six hatch styles).

4.1.5 Color

For plotters, color indices are mapped to the corresponding pen stations. For example, color index 1 is mapped to pen station one, color index 2 is mapped to pen station two, and so on. Color index 0 is not displayed.

For printers, refer to the device driver information sheet for specific information regarding color.

4.1.6 Alpha Text

Alpha text can be displayed anywhere on the page. The following text capabilities are available on plotters and printers:

- Fonts: 1= Normal (default)
2= Bold
- Sizes: 1 (default = 66lines).
- Color.
- Superscript and Subscript.
- Overstrike Mode.
- Underlining.

4.1.7 Graphics Text

Plotters support continuous character scaling. This text can be rotated in increments of a degree.

Printers support 0, 90, 180 and 270 degree rotation.

4.1.8 Paper Handling

On plotters the user can be prompted to change paper. On printers the output is generated at the Clear Workstation, Close Workstation and Update Workstation functions.

4.2 Apple LaserWriter

Filename

laserwriter

Device Logical Name

Plotter

Communications

This device must be directly connected to a serial tty. The LaserWriter cannot operate through the XENIX spooler.

Environmental Settings

The SCO CGI driver for the Apple LaserWriter is supplied with a *cgiprep* file containing PostScript code, which is downloaded into the printer by the device driver during Open Workstation. The file normally exists in the same directory as the *laserwriter* driver. If it has been moved, then the environmental variable **CGIPREP** must be set to the pathname of this file.

For example, if *cgiprep* was located in */usr/laser*, the **CGIPREP** variable could be set from the Bourne Shell (**sh**) or Korn Shell (**ksh**) in the following manner:

```
CGIPREP=/usr/laser/cgiprep
export CGIPREP
```

If you are using the Berkeley C-Shell (**csh**), use the following:

```
setenv CGIPREP /usr/laser/cgiprep
```

The following environmental settings can be specified as options:

1. Set Orientation to Portrait

With Portrait orientation, the output is printed with the paper being longer in the vertical (up/down) direction. This is the default setting.

If you are in the `sh` shell, set the environment by entering:

```
ORIENTATION=PORTRAIT
export ORIENTATION
```

If you are in the `cs` shell, set the environment by entering:

```
setenv ORIENTATION PORTRAIT
```

2. Set Orientation to Landscape

With this setting, output is rotated 270 degrees counterclockwise.

If you are in the `sh` shell, set the environment by entering:

```
ORIENTATION=LANDSCAPE
export ORIENTATION
```

If you are in the `cs` shell, set the environment by entering:

```
setenv ORIENTATION LANDSCAPE
```

3. Set Error Reporting to On

With this setting, messages from the printer, such as "Out of Paper" will be printed on the controlling terminal. Note that this is not the default setting.

If you are in the `sh` shell, set the environment by entering:

```
ERRORRPT=ON
export ERRORRPT
```

If you are in the `cs` shell, set the environment by entering:

```
setenv ERRORRPT ON
```

4. Set Error Reporting to Off

With this setting, messages from the printer are not printed. This is the default setting.

If you are in the `sh` shell, set the environment by entering:

```
ERRORRPT=OFF
export ERRORRPT
```

If you are in the `csh` shell, set the environment by entering:

```
setenv ERRORRPT OFF
```

Features Supported

Color

The LaserWriter supports two colors: Index 1 is displayed in black ink and index 0 is not displayed. These colors cannot be redefined.

Alpha Text

The LaserWriter supports alpha text in the following fonts and sizes, available in both Landscape and Portrait orientations:

```
Fonts: 1 = Courier
       2 = Courier Bold
       3 = Courier Oblique
       4 = Times Roman
       5 = Times Bold
       6 = Times Italic
       7 = Helvetica
       8 = Helvetica Bold
       9 = Helvetica Oblique
```

Sizes: 67 sizes of each font are supported, ranging from 6 points through 72 points in size. Size 1 selects 6-point text; size index 67 selects 72-point text.

Pass-through mode is not supported and alpha text quality is always 100.

4.3 Epson80Series Printers

Filename

epson80

Device Logical Name

Printer

Communications

This driver can use the XENIX spooler or can be connected directly.

Environmental Settings

The user can specify the following environmental settings:

1. Set Orientation to Portrait

This is the default setting. The output is not rotated.

If you are in the **sh** shell, set the environment by entering:

```
ORIENTATION=PORTRAIT
export ORIENTATION
```

If you are in the **cs**h shell, set the environment by entering:

```
setenv ORIENTATION PORTRAIT
```

2. Set Orientation to Landscape

With this setting, output is rotated 270 degrees counterclockwise.

If you are in the **sh** shell, set the environment by entering:

```
ORIENTATION=LANDSCAPE
export ORIENTATION
```

If you are in the **cs**h shell, set the environment by entering:

```
setenv ORIENTATION LANDSCAPE
```

3. Set Temporary Directory

TEMPDIR specifies the directory in which the printer driver writes any temporary files. The default directory is **/tmp**.

If you are in the **sh** shell, set the environment by entering:

```
TEMPDIR=directory_path
export TEMPDIR
```

If you are in the **csh** shell, set the environment by entering:

```
setenv TEMPDIR directory_path
```

4. Set DisplayList Size

PLISTSIZE informs the printer driver of the maximum number of bytes that the display list buffer accepts before off-loading the bytes to disk. The default and minimum size is 8K bytes. Increasing this value causes the specified number of bytes to be allocated at Open Workstation. It is intended for applications which can afford the memory to obtain increased performance. The maximum size is 64K bytes. These bytes are not released until the Close Workstation routine is invoked.

If you are in the **sh** shell, set the environment by entering:

```
PLISTSIZE=display_list_size
export PLISTSIZE
```

If you are in the **csh** shell, set the environment by entering:

```
setenv PLISTSIZE display_list_size
```

Features Supported

Color

The Epson supports two colors: Index 1 is displayed in black ink and index 0 is not displayed. These colors cannot be redefined.

Alpha Text

In addition to the common features, the following text capabilities are available on this printer:

Fonts: 1 = Normal (default)
2 = Bold
3 = Italics
7 = Bold Italics

Sizes:

Fonts 1 and 3

1 = 17 characters per inch
2 = 10 characters per inch (default)
3 = 8.5 characters per inch
4 = 5 characters per inch

Fonts 2 and 7

1 = 10 characters per inch (default)
2 = 5 characters per inch

Alpha text is only available in portrait mode.

4.4 Epson 100 Series Printers

Filename

epson100

Device Logical Name

Printer

Communications

This driver can use the XENIX spooler or can be directly connected.

Environmental Settings

The user can specify the following environmental settings:

1. Set Orientation to Portrait

If you are in the `sh` shell, set the environment by entering:

```
ORIENTATION=PORTRAIT
export ORIENTATION
```

If you are in the `csh` shell, set the environment by entering:

```
setenv ORIENTATIONPORTRAIT
```

2. Set Orientation to Landscape

If you are in the `sh` shell, set the environment by entering:

```
ORIENTATION=LANDSCAPE
export ORIENTATION
```

If you are in the `csh` shell, set the environment by entering:

```
setenv ORIENTATIONLANDSCAPE
```

3. Set Temporary Directory

`TEMPDIR` specifies the directory in which the printer driver writes any temporary files. The default directory is `/tmp`.

If you are in the **sh** shell, set the environment by entering:

```
TEMPDIR=directory_path
export TEMPDIR
```

If you are in the **cs**h shell, set the environment by entering:

```
setenv TEMPDIR directory_path
```

4. Set Display List Size

PLISTSIZE informs the printer driver of the maximum number of bytes that the display list buffer accepts before off-loading the bytes to disk. The default and minimum size is 8K bytes. Increasing this value causes the specified number of bytes to be allocated at Open Workstation. It is intended for applications which can afford the memory to obtain increased performance. The maximum size is 64K bytes. These bytes are not released until the Close Workstation routine is invoked.

If you are in the **sh** shell, set the environment by entering:

```
PLISTSIZE=display_list_size
export PLISTSIZE
```

If you are in the **cs**h shell, set the environment by entering:

```
setenv PLISTSIZE display_list_size
```

5. Set Paper Width to Narrow

This setting is for 8.5" by 11" graphics output. It is the default paper width.

If you are in the **sh** shell, set the environment by entering:

```
PAPER=NARROW
export PAPER
```

If you are in the **cs**h shell, set the environment by entering:

```
setenv PAPER NARROW
```

6. Set Paper Width to Wide

This setting is for 11" by 14" graphics output.

If you are in the `sh` shell, set the environment by entering:

```
PAPER=WIDE
export PAPER
```

If you are in the `csh` shell, set the environment by entering:

```
setenv PAPER WIDE
```

Features Supported

Color

The Epson supports two colors: Index 1 is displayed in black ink and index 0 is not displayed. These colors cannot be redefined.

Alpha Text

In addition to the common features, the following text capabilities are available on this printer:

Fonts: 1 = Normal (default)

2 = Bold

3 = Italics

7 = Bold Italics

Sizes:

Fonts 1 and 3

1 = 17 characters per inch

2 = 10 characters per inch (default)

3 = 8.5 characters per inch

4 = 5 characters per inch

Fonts 2 and 7

1 = 10 characters per inch (default)

2 = 5 characters per inch

Alpha text is not available under the following conditions:

- Environmental variable `PAPER` set to `NARROW`, in landscape mode
- Environmental variable `PAPER` set to `WIDE`, in portrait mode.

Special Considerations

For PAPER set to WIDE: default is Landscape mode.
Output in Portrait mode is rotated 270 degrees counterclockwise.

For PAPER set to NARROW: default is Portrait mode.
Output in Landscape mode is rotated 270 degrees counterclockwise.

4.5 Hewlett-Packard Plotters

Filename

hpplot

Device Logical Name

Plotter

Communications

This device must be directly connected to a serial tty. The Hewlett-Packard Plotters cannot operate through the XENIX spooler.

Environmental Settings

The user can specify the following environmental settings:

1. Set Orientation to Portrait

Output is not rotated. The default is landscape mode.

If you are in the `sh` shell, set the environment by entering:

```
ORIENTATION=PORTRAIT
export ORIENTATION
```

If you are in the `cs` shell, set the environment by entering:

```
setenv ORIENTATION PORTRAIT
```

2. Set Orientation to Landscape

With this setting, output is rotated 270 degrees counterclockwise. This setting is the default.

If you are in the `sh` shell, set the environment by entering:

```
ORIENTATION=LANDSCAPE
export ORIENTATION
```

If you are in the `cs` shell, set the environment by entering:

```
setenv ORIENTATION LANDSCAPE
```

The plotter driver determines which model it is driving (models 7440, 7470A, 7475A, 7550, 7580, 7585, or 7586 are supported). This driver will also support IBM 6180, 7371, 7372, 7374, 7375-1, and 7375-2 plotters.

Features Supported

Request Locator

The pen holder is used to indicate the point to be selected. Move the pen holder by pressing the position keys on the front panel. When the pen holder is positioned correctly, press the ENTER button. This transmits the coordinates of the point back to the user program.

Alpha Text

In addition to the common features, the following text capability is available on this plotter:

Fonts: 1 = Normal
2 = Bold
3 = Italics

4.6 Hewlett-Packard Laserjet Printer

Filename

laserjet

Device Logical Name

Printer

Communications

This driver can use the XENIX spooler or can be connected directly.

Environmental Settings

The user can specify the following environmental settings:

1. Set Orientation to Portrait

This is the default setting. The output is not rotated.

If you are in the **sh** shell, set the environment by entering:

```
ORIENTATION=PORTRAIT
export ORIENTATION
```

If you are in the **csh** shell, set the environment by entering:

```
setenv ORIENTATION PORTRAIT
```

2. Set Orientation to Landscape

The output is rotated 270 degrees counterclockwise.

If you are in the **sh** shell, set the environment by entering:

```
ORIENTATION=LANDSCAPE
export ORIENTATION
```

If you are in the **csh** shell, set the environment by entering:

```
setenv ORIENTATION LANDSCAPE
```

3. Set Verifyfont to On

This is the default setting.

With this setting, only software graphic text fonts whose x and y pixel sizes match the driver's pixel sizes are available to the application. Candidate font files are listed in the **fontlist.dat** file.

If you are in the **sh** shell, set the environment by entering:

```
VERIFYFONT=ON
export VERIFYFONT
```

If you are in the **cs**h shell, set the environment by entering:

```
setenv VERIFYFONT ON
```

4. Set Verifyfont to Off

With this setting, all software graphic text fonts listed in the **fontlist.dat** file are available to the application.

If you are in the **sh** shell, set the environment by entering:

```
VERIFYFONT=OFF
export VERIFYFONT
```

If you are in the **cs**h shell, set the environment by entering:

```
setenv VERIFYFONT OFF
```

5. Set Font Cartridge

CARTRIDGE tells the driver the one letter name, A-Z, of the font cartridge installed in the printer.

If you are in the **sh** shell, set the environment by entering:

```
CARTRIDGE=capital_letter
export CARTRIDGE
```

If you are in the **cs**h shell, set the environment by entering:

```
setenv VERIFYFONT capital_letter
```

6. Set Printer Resolution

The default setting is 150 dots per inch.

The **laserjet** has the ability to print graphics at 75, 100, and 150 dots per inch.

If you are in the **sh** shell, set the environment by entering:

```
RESOLUTION=selected_resolution
export RESOLUTION
```

If you are in the **cs**h shell, set the environment by entering:

```
setenv RESOLUTION selected_resolution
```

7. Set Temporary Directory

TEMPDIR specifies the directory in which the printer driver writes any temporary files. The default directory is **/tmp**.

If you are in the **sh** shell, set the environment by entering:

```
TEMPDIR=directory_path
export TEMPDIR
```

If you are in the **cs**h shell, set the environment by entering:

```
setenv TEMPDIR directory_path
```

8. Set Display List Size

PLISTSIZE informs the printer driver of the maximum number of bytes that the display list buffer accepts before off-loading the bytes to disk. The default and minimum size is 8K bytes. Increasing this value causes the specified number of bytes to be allocated at Open Workstation. It is intended for applications which can afford the memory to obtain increased performance. The maximum size is 64K bytes. These bytes are not released until the Close Workstation routine is invoked.

If you are in the **sh** shell, set the environment by entering:

```
PLISTSIZE=display_list_size  
export PLISTSIZE
```

If you are in the **csh** shell, set the environment by entering:

```
setenv PLISTSIZE display_list_size
```

Features Supported

Color

The Laserjet supports two colors: Index 1 is displayed in black ink and index 0 is not displayed. These colors cannot be redefined.

Alpha Text

In addition to the common features, the following text capabilities are available on this printer:

Fonts: 1 = Normal (default)
2 = Bold

Sizes: 1 = 10 characters per inch

Alpha text is only available in portrait mode.

4.7 Hewlett-Packard ThinkjetPrinter

Filename

thinkjet

Device Logical Name

Printer

Communications

This driver can use the XENIX spooler or can be connected directly.

Environmental Settings

The user can specify the following environmental settings:

1. Set Orientation to Portrait

This is the default setting. The output is not rotated.

If you are in the `sh` shell, set the environment by entering:

```
ORIENTATION=PORTRAIT
export ORIENTATION
```

If you are in the `cs` shell, set the environment by entering:

```
setenv ORIENTATION PORTRAIT
```

2. Set Orientation to Landscape

The output is rotated 270 degrees counterclockwise.

If you are in the `sh` shell, set the environment by entering:

```
ORIENTATION=LANDSCAPE
export ORIENTATION
```

If you are in the `cs` shell, set the environment by entering:

```
setenv ORIENTATION LANDSCAPE
```

3. Set Temporary Directory

TEMPDIR specifies the directory in which the printer driver writes any temporary files. The default directory is */tmp*.

If you are in the *sh* shell, set the environment by entering:

```
TEMPDIR=directory_path
export TEMPDIR
```

If you are in the *csh* shell, set the environment by entering:

```
setenv TEMPDIR directory_path
```

4. Set Display List Size

PLISTSIZE informs the printer driver of the maximum number of bytes that the display list buffer accepts before off-loading the bytes to disk. The default and minimum size is 8K bytes. Increasing this value causes the specified number of bytes to be allocated at Open Workstation. It is intended for applications which can afford the memory to obtain increased performance. The maximum size is 64K bytes. These bytes are not released until the Close Workstation routine is invoked.

If you are in the *sh* shell, set the environment by entering:

```
PLISTSIZE=display_list_size
export PLISTSIZE
```

If you are in the *csh* shell, set the environment by entering:

```
setenv PLISTSIZE display_list_size
```

Features Supported

Color

The Thinkjet supports two colors: Index 1 is displayed in black ink and index 0 is not displayed. These colors cannot be redefined.

Alpha Text

In addition to the common features, the following text capabilities are available on this printer:

Fonts: 1 = Normal (default)
2 = Bold

Sizes: 1 = 21.3 characters per inch
2 = 12 characters per inch (default)
3 = 10.7 characters per inch
4 = 6 characters per inch

Alpha Text Quality:
50 : Turn on bi-direction printing
50 : Turn off bi-direction printing

Alpha text is only available in portrait mode.

Chapter 5

Graphics Input Devices

- 5.1 Features Common To Graphics Input Devices 5-1
 - 5.1.1 SetInputExtent 5-1
 - 5.1.2 RequestLocator 5-1
 - 5.1.3 SampleLocator 5-1



5.1 Features Common To Graphics Input Devices

Graphics input device drivers are input-only drivers and must be used in conjunction with an output echo display device driver.

5.1.1 SetInputExtent

The input extent rectangle controls the range of coordinates returned by the REQUEST LOCATOR and SAMPLE LOCATOR functions. When the application sets the input extent rectangle of the graphics input device, the coordinates of all points returned by REQUEST LOCATOR and SAMPLE LOCATOR are transformed so that they are within the specified rectangle.

5.1.2 RequestLocator

When the REQUEST LOCATOR function is invoked, a graphics input tracking cursor appears on the output echo device at the initial locator position. The graphics input cursor can be moved by the input device.

When the graphics input cursor is at the desired location, the point can be selected by pressing a button or key. This causes the coordinates of the point to be transmitted back to the user program, along with an ASCII character code for the button pressed. Refer to the specific device driver information sheet for the character codes associated with the buttons.

5.1.3 Sample Locator

When the SAMPLE LOCATOR function is invoked, the graphics input device returns the current position and key state information without waiting for operator interaction.

The key state is returned in three 16-bit integers: the current state of the mouse buttons, the keys pressed since the last input inquiry, and the keys released since the last inquiry. Each button is represented by a single bit in each of the 16-bit integers returned to the application. The minimum key state is zero (no buttons pushed), and the maximum key state is $2^n - 1$, where n is the number of buttons available.

SCO CGI Device Driver Supplement

May 25, 1987

No Graphics Input device drivers are currently available.

Chapter 6

Other Types of Devices

6.1 AboutTheseDevices 6-1

6.2 ComputerGraphicsMetafile(CGM) 6-2

6.3 GSS*GRAFSTATION 6-4

6.1 About These Devices

This section is reserved for device drivers that increase the capabilities of SCO CGI. Because they are distinctive, they are not appropriate for other sections.

An example is the Computer Graphics Metafile, or CGM, driver. This driver manages metafiles, the computer graphics picture exchange and storage medium. The CGM driver is not a display, hard copy, or a graphics input driver. It is a special driver.

It is not possible to describe the common features of these devices; each one is unique.

6.2 Computer Graphics Metafile (CGM)

Filename

ddmeta

Device Logical Name

METAFIL

Communications

not applicable

Environmental Settings

The default filename is **metafile.dat**.

To change the default, if you are in the **sh** shell, set the environment by entering:

```
METAOUTPUT=filename
export METAOUTPUT
```

If you are in the **csh** shell, set the environment by entering:

```
setenv METAOUTPUT filename
```

Specific Features Supported

Polylines

The metafile driver supports 32767 line widths and styles.

Polymarkers

The metafile supports 32767 styles and sizes of polymarkers.

Graphics Text

The metafile supports 32767 fonts, sizes and rotations.

Color

The metafile driver supports 256 color indices with 1000 variations of red, green and blue. The default colors are:

Index	Color
0	black
1	white
2	red
3	green
4	blue
5	yellow
6	cyan
7	magenta
8-255	white

A user can change a color by selecting desired levels of the three color components that make up the index: red, green and blue. This method can be used to create nondefault colors such as brown or orange. The new color will be visible only on color devices that allow color definition.

6.3 GSS*GRAFSTATION

Filename

gstd

Device Logical Name

DISPLAY

Communications

not applicable

Features Supported

The GSS*GRAFSTATION driver forms an interface between the host computer and a GSS*GRAFSTATION device. This interface is capable of passing information about any of the features that are supported by SCO CGI. Thus, the features that are supported by this device driver are limited only by the set of features supported by the GSS*GRAFSTATION device used, and its graphics peripherals.

The GSS*GRAFSTATION driver is part of SCO CGI. However, in order for this device driver to operate properly, additional software must be installed on your output device. The results obtained from using the GSS*GRAFSTATION device driver will vary according to the output device you are using.

Glossary

10

11

12

- ADE** ASCII Decimal Equivalents; decimal numbers used in code to represent ASCII characters. For example, 65=A and 66=B. ADE character parameters are passed and returned as integers.
- argument** One of the independent variables that the action or output of a routine depends on. Arguments are enclosed in parentheses in the routine call.
- array** Series of related items (data) arranged in a meaningful pattern.
- aspects of primitives** Ways in which the appearance of a primitive can vary. Aspects are controlled directly by primitive attributes.
- attribute functions** Primitive attributes affect the appearance of objects created with primitive routines (for example, character height or line style).
- binding** A language binding is the specification of the exact calling syntax and data types for arguments to be used when calling SCO CGI routines from a specific programming language.
- CGI** See *Computer Graphics Interface*.
- Cartesian coordinate system**
Coordinate system composed of an x axis (horizontal) increasing positively towards the right and a y axis (vertical) increasing positively upwards. The axes are positioned at right angles, and the point of intersection is the origin (0,0). The position of any point is defined by the displacement from the origin along the x and the y axes.
- cell array** SCO CGI output primitive consisting of a rectangular grid of equal size rectangular cells, each having a single color. These cells may not map one-to-one with frame buffer pixels.
- choice input device** A logical input device that offers the user a set of alternatives and returns an integer value indication of the option selected. An example device is a set of function keys.
- clipping** If the visible space is smaller than the virtual coordinate space, CGI must make those portions of the object outside the visible space

invisible. The clip rectangle determines which portion of the virtual coordinate space is visible. If the coordinates outside the visible space were not clipped, the image displayed would vary from device to device.

- color map Table designed to provide a range of colors by defining different mixtures of the RGB color components. A desired color is referenced by its assigned number. The identifying numbers with their assigned colors are called the color map. Changing colors assigned to the identifying number changes the color map.
- color table Workstation-dependent table in which the entries specify the values of the red, green and blue intensities defining a particular color.

Computer Graphics Interface

The Computer Graphics Interface (CGI) is a standard interface between device-dependent and device-independent code in a graphics environment. CGI makes all device drivers appear identical to the calling program. SCO CGI is based on CGI and all device drivers written for SCO CGI must conform to the CGI specification.

- control functions These functions allow you to exercise control over certain aspects of the system and the display device.
- coordinate scaling Coordinate scaling transforms points from one *space* to another. In SCO CGI all point coordinates must be specified in Virtual Device Coordinates with values between -32768 and 32767. These coordinates are then scaled into values which are appropriate for your graphics device.
- default A value assigned to a parameter by SCO CGI and used when you do not specify a value.
- device coordinate A coordinate expressed in a coordinate system that is device-dependent.
- device driver Device-dependent software that communicates with a specific graphics device to draw graphics on the display surface based on SCO CGI functions.

device handle	Number returned that identifies a unique device.
device-independent	The ability to be used on more than one type of graphics display device.
device space	The space defined by the addressable points of a graphics device.
display device	A device (for example, a refresh display or a storage tube display) on which display images can be represented.
display surface	In a display device, that medium on which display images may appear.
echo	The immediate notification of the current value provided by an input device to the operator at the display console.
escape	A routine within SCO CGI that is the only access to implementation-dependent or device-dependent support for nonstandard functionality other than graphics output.
fill area	An SCO CGI output primitive consisting of a polygon (closed boundary) that may be hollow or may be filled with a uniform color, a pattern, or a hatch style.
Generalized Drawing Primitive	
	A display element (output primitive) used to address special geometrical workstation capabilities such as curve drawing.
GKS	See <i>Graphical Kernel System</i> .
Graphical Kernel System	
	The Graphical Kernel System (GKS) is an international standard for the programmer's interface to graphics.
graphics primitives	Graphics primitives are the basic graphics operations performed by SCO CGI (for example, drawing lines, markers, and text strings).
host-independent	Capable of running on a number of operating systems.

hot spot	The point of a marker or cursor used to specify its position on the display surface. In the case of a cross, an appropriate <i>hot spot</i> might be its center. In the case of an arrow, an appropriate <i>hot spot</i> might be its tip.
input functions	The Input Functions return information from the operator. These functions return the point location from the input device, return the status of a valuator device, return the status of a choice device, and allow input from the keyboard. They operate in sample and request modes.
inquiry functions	The Inquiry Functions allow your program to determine the present state of the system. You can determine the current value of the following: <ul style="list-style-type: none">● primitive attributes● device capabilities● device state● error state
landscape mode	One of the two possible paper orientations for hardcopy devices. In landscape mode, the <i>x</i> measurement is greater than the <i>y</i> measurement. Compare <i>portrait mode</i> .
locator input device	An SCO CGI logical input device providing a position in virtual device coordinates. For example, a mouse is a locator input device.
null-terminated string	A one-dimensional array or list of characters. The end of a string is indicated by the NULL character (ADE0).
output primitives	Functions that cause graphics to be drawn on a display surface. These functions describe polylines, polymarkers, text strings, pixel arrays, fill areas and generalized drawing primitives. The invocation of an output primitive function results in an output primitive, such as a sequence of markers or polylines. The appearance of output primitives is affected by the values of primitive attributes.

- pixel** The smallest element of a display surface that can be independently assigned a color or intensity.
- polyline** An SCO CGI output primitive consisting of a set of connected lines.
- polymarker** An SCO CGI output primitive consisting of a set of locations to be indicated by a marker.
- portrait mode** One of the two possible paper orientations for hardcopy devices. In portrait mode, they measurement is greater than the *x* measurement. Compare *landscape mode*.
- raster** A field of closely spaced lines on the face of a video terminal that defines an image. The spacing between raster lines defines the resolution of a display.
- RGB** An additive method for defining color. In this color model, percentages of the primaries—red, green and blue—are added to produce colors:
- 100% red + 100% green + 100% blue = white.
 100% red + 100% green = yellow.
 100% red + 100% blue = magenta.
 100% blue + 100% green = cyan.
- SCO CGI** SCO CGI is a host- and device-independent graphics sub-system that serves as an environment for graphics applications as well as for application development.
- stringinput device** A logical input device that provides a text string; for example, a keyboard.
- transformation** The mapping of objects from one coordinate space to another; for example, from virtual device coordinates to device coordinates.
- valuator input device** A logical input device that returns scalar values in a range (0 to 32767) that is proportional to the valuator position; for example, a control dial.
- VDC** See *Virtual Device Coordinate*.
- VirtualDevice Coordinate**
 Virtual Device Coordinates (VDCs) form a space in which the full extent of the device axes

are assigned values between -32768 and 32767. This convention provides improved device independence for a graphics system by allowing the viewing operations to be carried out without regard for device-coordinate specifics. The *VDC Coordinates* are transformed to specific device coordinates by SCO CGI .

workstation

SCO CGI is based on the concept of abstract graphics workstations that provide the logical interface through which the application program controls one or more input/output devices.

SCO XENIX®

Development System

C Language Reference Guide



Information in this document is subject to change without notice and does not represent a commitment on the part of The Santa Cruz Operation, Inc. nor Microsoft Corporation. The software described in this document is furnished under a license agreement or non-disclosure agreement. The software may be used or copied only in accordance with the terms of the agreement. It is against the law to copy this software on magnetic tape, disk, or any other medium for any purpose other than the purchaser's personal use.

Portions © 1986, 1987 Graphic Software Systems, Inc.
All rights reserved.

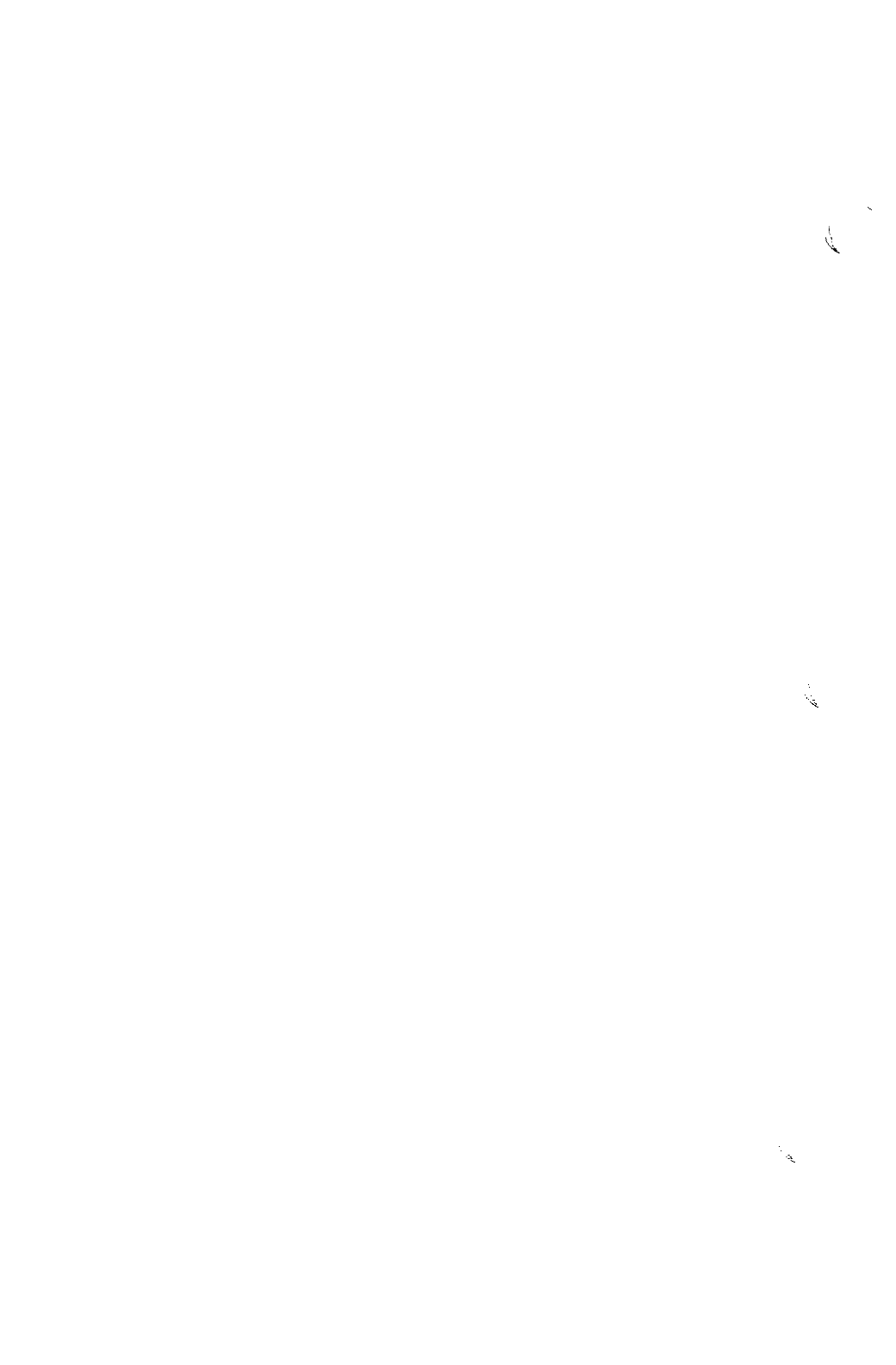
Portions © 1987 The Santa Cruz Operation, Inc.
All rights reserved.

ALL USE, DUPLICATION, OR DISCLOSURE WHATSOEVER BY THE GOVERNMENT SHALL BE EXPRESSLY SUBJECT TO RESTRICTIONS AS SET FORTH IN SUBDIVISION (b) (3) (ii) FOR RESTRICTED RIGHTS IN COMPUTER SOFTWARE AND SUBDIVISION (b) (2) FOR LIMITED RIGHTS IN TECHNICAL DATA, BOTH AS SET FORTH IN FAR 52.227-7013.

This document was typeset with an IMAGEN® 8/300 Laser Printer.

Microsoft and XENIX are registered trademarks of Microsoft Corporation.
IMAGEN is a registered trademark of IMAGEN Corporation.

SCODocumentNumber: XG-6-12-87-1.0
Processed: Wed Jun 10 14:47:28 PDT 1987



Contents

1 Introduction

- 1.1 Overview 1-1
- 1.2 Conventions Used In This Guide 1-1

2 Compilation and Linking Procedures

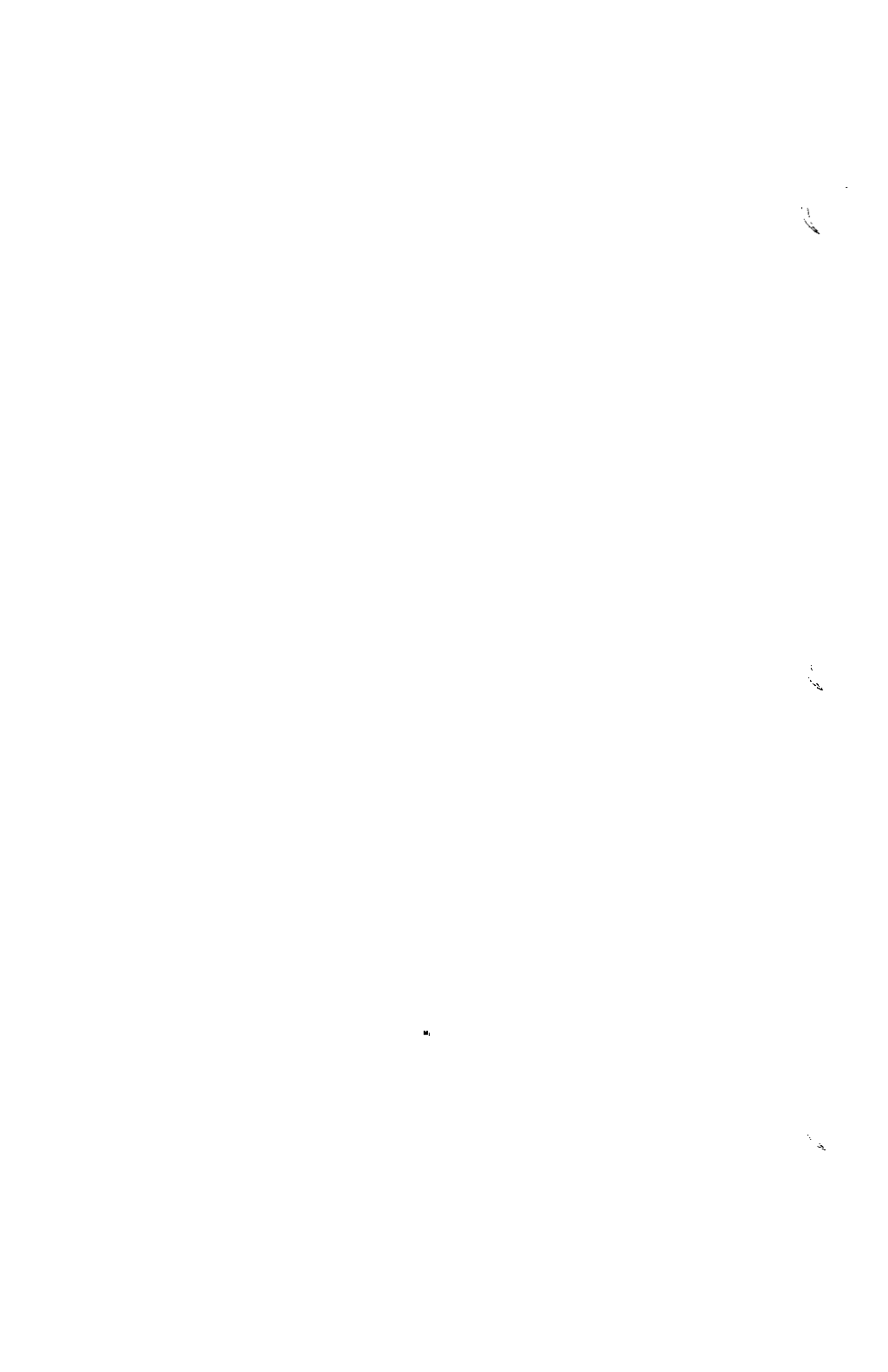
- 2.1 Procedure For XENIX 2-1

3 Master Function List

- 3.1 Overview 3-1
- 3.2 Control Functions 3-1
- 3.3 Bitmap Functions 3-2
- 3.4 Output Functions 3-2
- 3.5 Attribute Functions 3-3
- 3.6 Input Functions 3-5
- 3.7 Inquiry Functions 3-5
- 3.8 File Input/Output Functions 3-7

4 SCO CGI Functions

- 4.1 Overview 4-1
- 4.2 The SCO CGI Functions 4-1



Chapter 1

Introduction

1.1 Overview 1-1

1.2 Conventions Used In This Guide 1-1



1.1 Overview

The language binding makes SCO CGI specific to a particular programming language. The binding is incorporated when you link your object files with SCO CGI as described in Chapter 2, **Compilation and Linking Procedures**.

This guide describes the C Language calling sequences for each of the SCO CGI functions. This guide also describes parameter passing and other conventions that are unique to C Language. Use the information in this guide as you write your source code. Refer to the *SCO CGI Programmer's Guide* for generic descriptions of the arguments and operations.

This guide is organized as follows.

Chapter 2	Compilation and Linking Procedures - C Language specific commands for compiling and linking;
Chapter 3	Master Function List - complete list of the SCO CGI functions, organized by their classes and sub-classes;
Chapter 4	SCO CGI Functions - C Language specific calling sequences and complete parameter list for each of the SCO CGI functions, arranged alphabetically by their generic function names.

In addition to this guide, the following SCO CGI publications are also available:

- *SCO CGI Programmer's Guide* - detailed generic descriptions of the SCO CGI functions and installation instructions;
- *SCO CGI Device Driver Supplement* - device specific data.

1.2 Conventions Used In This Guide

The following conventions are used:

- SINT16 = a signed 16 bit integer;
- CHAR = a signed 8 bit integer used as a character;
- SINT32 = a signed 32 bit integer;
- BIT8 = an 8 bit value not used arithmetically;

SCO CGI C Language Reference Guide

- BIT16= a 16 bit value not used arithmetically;
- BIT32= a 32 bit value not used arithmetically;
- FD = a 16 bit value used as a file descriptor;
- Scalar input arguments are passed as value;
- Array input arguments are passed by address;
- All output arguments are passed by address;
- All inquiry procedures are functions that return an integer value;
- No other functions return a value.

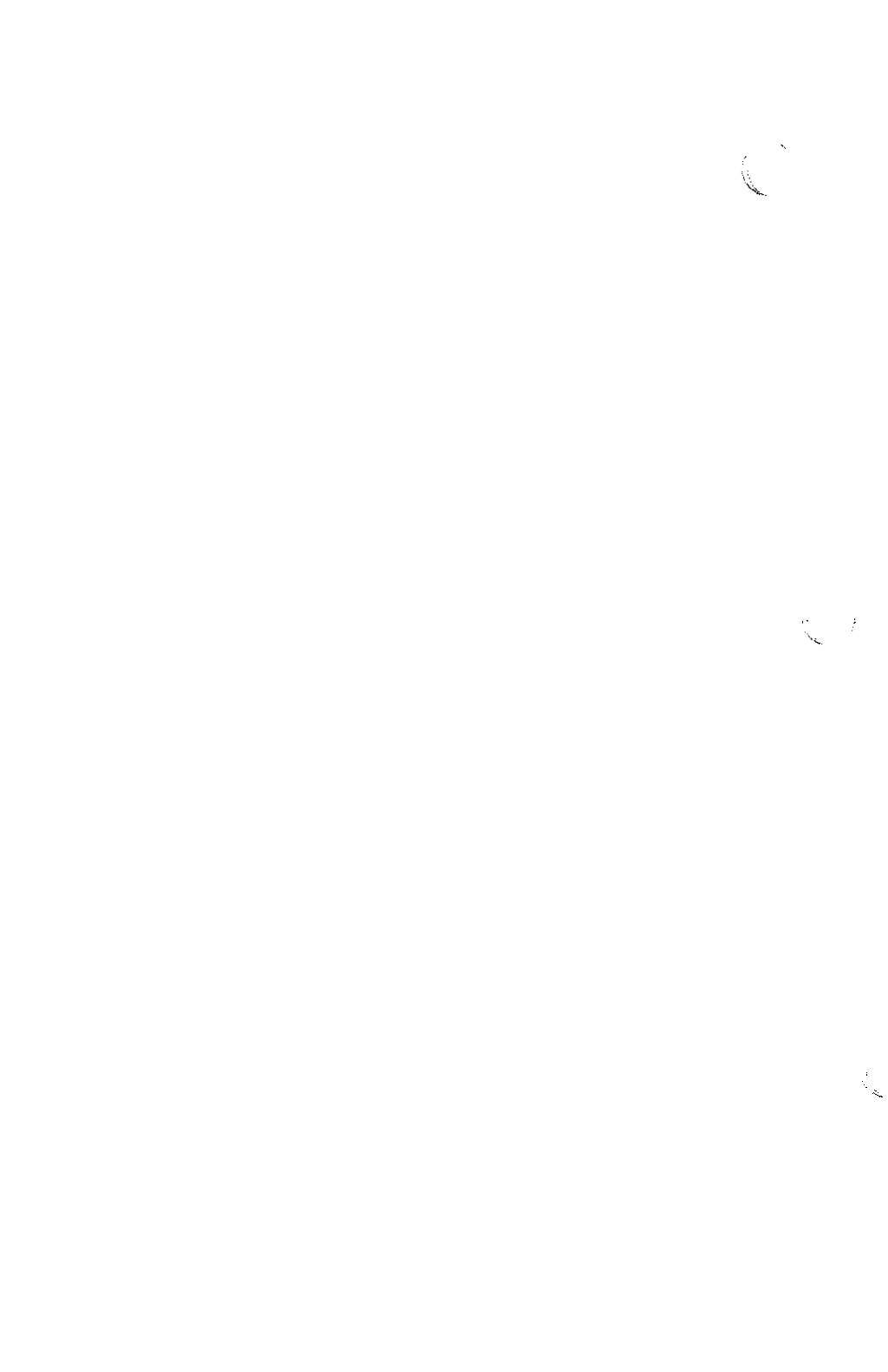
Output parameters are listed in *this italic type*. Input parameters are listed in regular type.

Chapter 2

Compilation and Linking

Procedures

2.1 Procedure For XENIX 2-1



2.1 Procedure For XENIX

To compile your source code, issue the following command.

```
cc <fname> -lccgi -o <fname>
```

where *<fname>* is your program name.

Note

This assumes that the C language binding library *libccgi.a* is in the proper directory so that your system's loader can find it using the *-l* (dash ell) switch.

The command listed above assumes that the distributed SCO library is called *ccgi*. Please check distribution diskettes to verify the library name.

1

1

1

1

1

1

1

1

1

1

1

1

1

1

1

1

1

Chapter 3

Master Function List

- 3.1 Overview 3-1
- 3.2 Control Functions 3-1
- 3.3 Bitmap Functions 3-2
- 3.4 Output Functions 3-2
- 3.5 Attribute Functions 3-3
- 3.6 Input Functions 3-5
- 3.7 Inquiry Functions 3-5
- 3.8 File Input/Output Functions 3-7



3.1 Overview

There are seven classes of CGI functions: Control, Bitmap, Output, Attribute, Input, Inquiry, and File Input/Output. The following tables list each CGI function by class. The functions are further divided into sub-classes for easy referencing. The generic function name is given next, then the C Language function name. Detailed descriptions of the functions are in the *SCO CGI Programmer's Guide*. Functions in the *Programmer's Guide* and this binding guide are ordered alphabetically according to their generic name.

3.2 Control Functions

Sub-Class	Generic Function Name	C Language
Alpha Text Control	Set Alpha Text Position	vsa_position
CGI Control	Load CGI Remove CGI Set Deferral Mode	cgi_load cgi_kill vs_defer
Cursor Text Control	Cursor Down Cursor Home Cursor Left Cursor Right Cursor Up Direct Cursor Address Enter Cursor Addressing Mode Erase to End of Line Erase to End of Screen Exit Cursor Addressing Mode	v_curdown v_curhome v_curleft v_curright v_curup vs_curaddress v_enter_cur v_eeol v_eeos v_exit_cur
Device Control	Hardcopy Set Background Mode Set Pen Speed Set Writing Mode	v_hardcopy vsb_mode vs_penspeed vswr_mode

Graphics Cursor Control	Create Cursor	vc_cursor
	Delete Cursor	vd_cursor
	Display Graphics	
	Input Cursor	v_dspcur
	Remove Graphics	
	Input Cursor	v_rmcur
Metafile	Select Graphics	
	Input Cursor	vsg_cursor
	Application Data Message	v_appl v_message
Work- station Control	Clear Workstation	v_clrwk
	Close Workstation	v_clswk
	Open Workstation	v_opnwk
	Reset To Defaults	vr_defaults
	Update Workstation	v_updwk

3.3 Bitmap Functions

<u>Generic Function Name</u>	<u>C Language</u>
Copy Bitmap	vcp_bitmap
Create Bitmap	vc_bitmap
Delete Bitmap	vd_bitmap
Select Drawing Bitmap	vsd_bitmap

3.4 Output Functions

<u>Generic Function Name</u>	<u>C Language</u>
Output Alpha Text	v_atext
Output Arc	v_arc
Output Bar	v_bar
Output Byte Pixel Array	vb_pixels
Output Cell Array	v_cellarray
Output CGI Error	v_perror
Output Circle	v_circle
Output Cursor	
Addressable Text	v_curtext
Output Ellipse	v_ellipse
Output Elliptical Arc	ve_arc
Output Elliptical Pie Slice	veL_pieslice
Output Filled Area	v_fillarea
Output Graphics Text	v_gtext
Output Integer Pixel Array	vi_pixels

OutputPieSlice	v_pieslice
Output Polyline	v_pline
Output Polymarker	v_pmarker

3.5 Attribute Functions

Sub- Class	Generic Function Name	C Language	
Alpha Text	Set Alpha Text Color Index	vsa_color	
	Set Alpha Text Font And Size	vsa_font	
	Set Alpha Text Line Spacing	vsa_spacing	
	Set Alpha Text Overstrike Mode	vsa_overstrike	
	Set Alpha Text Pass Through Mode	vsa_passthru	
	Set Alpha Text Quality	vsa_quality	
	Set Alpha Text Sub/Superscript Mode	vsa_supersub	
	Set Alpha Text Underline Mode	vsa_underline	
	Back-ground	Set Background Color Index	vsb_color
		Set Clip Rectangle	vsc_rectangle
	Color	Set Color Representation	vs_color
		Set Color Table	vsc_table
		Reverse Video Off	v_rvoff
Cursor Text	Reverse Video On	v_rvon	
	Set Cursor Text Attributes	vcur_att	
	Set Cursor Text Color Index	vcur_color	
	Fill	Set Fill Area Representation	vsf_representation
Set Fill Color Index		vsf_color	

SCO CGI C Language Reference Guide

	Set Fill Interior Style	vsf_interior
	Set Fill Style Index	vsf_style
Graphics Text	Set Graphics Text Alignment	vst_alignment
	Set Graphics Text Character Height	vst_height
	Set Graphics Text Color Index	vst_color
	Set Graphics Text Font	vst_font
	Set Graphics Text Representation	vst_representation
	Set Graphics Text String Baseline Rotation	vst_rotation
Input	Set Line Edit Characters	vs_editchars
Polyline	Set Polyline Color Index	vsl_color
	Set Line Cross Section	vsl_cross
	Set Line Type	vsl_type
	Set Line Width	vsl_width
	Set Polyline Representation	vsl_representation
	Set User Polyline Line Type	vsul_type
Polymarker	Set Polymarker Color Index	vsm_color
	Set Polymarker Height	vsm_height
	Set Polymarker Representation	vsm_representation
	Set Polymarker Type	vsm_type

3.6 Input Functions

Sub-Class	Generic Function Name	C Language
Choice	Request Choice	vrq_choice
	Sample Choice	vsm_choice
Cursor	Read Cursor Keys	vr_d_curkeys
Input Extent	Set Input Extent	vsi_extent
Locator	Request Locator	vrq_locator
	Sample Locator	vsm_locator
String	Request String	vrq_string
	Sample String	vsm_string
Valuator	Request Valuator	vrq_valuator
	Sample Valuator	vsm_valuator

3.7 Inquiry Functions

Sub-Class	Generic Function Name	C Language
Bitmap	Inquire Bitmap Formats	vqb_format
	Inquire Byte Pixel Array	vqb_pixels
	Inquire Displayable Bitmaps	vq_displays
	Inquire Drawing Bitmap	vqd_bitmap
	Inquire Integer Pixel Array	vqi_pixels
Clipping	Inquire Clip Rectangle	vqc_rectangle
Cursor	Inquire Cursor Description	vq_cursor

	Inquire Graphics Input Cursor	vqg_cursor
Device Capabilities	Inquire Addressable Character Cells	vq_chcells
Errors	Inquire CGIError	vq_error
File I/O	Inquire File Status	fd_inq
Input Extent	Inquire Input Extent	vqi_extent
Primitive	Inquire Back- ground Mode	vqb_mode
	Inquire Cell Array	vq_cellarray
	Inquire Color Representation	vq_color
	Inquire Current Fill Area Attributes	vqf_attributes
	Inquire Current Polyline Attributes	vql_attributes
	Inquire Current Polymarker Attributes	vqm_attributes
	Inquire Fill Area Representation	vqf_representation
	Inquire Optimum Pattern Size	vqo_pattern
	Inquire Polyline Representation	vql_representation
	Inquire Polymarker Representation	vqm_representation
Text	Inquire Alpha Text Capabilities	vqa_cap
	Inquire Alpha Text Cell Location	vqa_cell
	Inquire Alpha Text Font Capability	vqa_font
	Inquire Alpha Text Position	vqa_position

Inquire Alpha Text String Length	vqa_length
Inquire Current Cursor Text Address	vq_curaddress
Inquire Current Graphics Text Attributes	vqt_attributes
Inquire Graphics Text Extent	vqt_extent
Inquire Graphics Text Font Character	vqtf_character
Inquire Graphics Text Font Description	vqtf_description
Inquire Graphics Text Font Metrics	vqtf_metrics
Inquire Graphics Text Representation	vqt_representation

3.8 FileInput/OutputFunctions

<u>Generic Function Name</u>	<u>C Language</u>
Close File	fd_close
Connect Directory Name	fd_connect
Copy Directory Name	fd_copy
Delete File	fd_delete
Disconnect Directory Name	fd_disconnect
File Size	fd_size
Open File	fd_open
Parse File Name	fd_parse
Read Directory	fd_directory
Read File Proceed	fdp_read
Read File Wait	fd_read
Rename File	fd_rename
Seek File	fd_seek
Write File Proceed	fdp_write
Write File Wait	fd_write

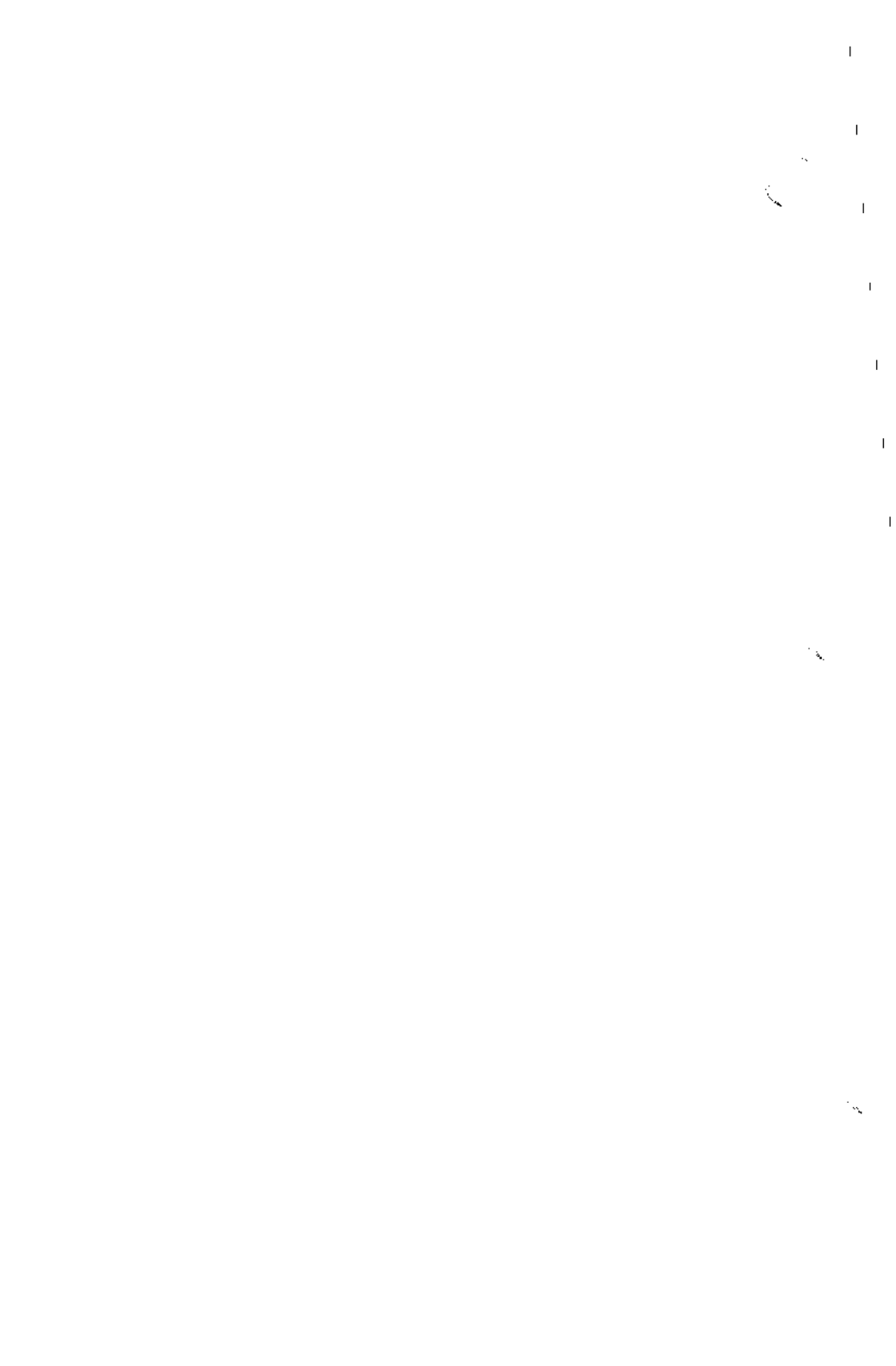


Chapter 4

SCO CGI Functions

4.1 Overview 4-1

4.2 The SCO CGI Functions 4-1



4.1 Overview

This chapter lists the C Language specific calling sequences and their parameters for each of the SCO CGI functions.

Output parameters are listed in *this italic type*. Input parameters are listed in regular type.

4.2 The SCO CGI Functions

Application Data

SINT16v_appl(dev_handle,function, data_cnt, app_data)

```
SINT16 dev_handle;  
CHAR function[];  
SINT16 data_cnt;  
SINT16 app_data[];
```

Clear Workstation

SINT16v_clrwk (dev_handle)

```
SINT16 dev_handle;
```

Close File

SINT16fd_close (fd)

```
FD fd;
```

Close Workstation

SINT16v_clswk (dev_handle)

```
SINT16 dev_handle;
```

Connect Directory Name

FD fd_connect (directory)

CHAR directory[];

Copy Bitmap

SINT16 vcp_bitmap (dev_handle, bitmap_handle, source_rect, dest_orig)

SINT16 dev_handle;
SINT16 bitmap_handle;
SINT16 source_rect[4];
SINT16 dest_orig[2];

Copy Directory Name

FD fd_copy (fd)

FD fd;

Create Bitmap

SINT16 vc_bitmap (dev_handle, xy, type, bitmap_handle)

SINT16 dev_handle;
SINT16 xy[4];
SINT16 type;
SINT16 bitmap_handle;*

Create Cursor

SINT16 vc_cursor (dev_handle, data_bitmap, mask_bitmap, x, y, cursor_handle)

SINT16 dev_handle;
SINT16 data_bitmap;
SINT16 mask_bitmap;
SINT16 x;
SINT16 y;
SINT16 cursor_handle;*

Cursor Down

SINT16v_curdown (dev_handle)

SINT16 dev_handle;

Cursor Home

SINT16v_curhome (dev_handle)

SINT16 dev_handle;

Cursor Left

SINT16v_curleft (dev_handle)

SINT16 dev_handle;

Cursor Right

SINT16v_currigh (dev_handle)

SINT16 dev_handle;

Cursor Up

SINT16v_curup (dev_handle)

SINT16 dev_handle;

Delete Bitmap

SINT16vd_bitmap (dev_handle, bitmap_handle)

SINT16 dev_handle;
SINT16 bitmap_handle;

Delete Cursor

SINT16vd_cursor(dev_handle, cursor_handle)

SINT16 dev_handle;
SINT16 cursor_handle;

Delete File

SINT16fd_delete(fd,name)

FDfd;
CHARname[];

Direct Cursor Address

SINT16vs_curaddress(dev_handle,row, column)

SINT16 dev_handle;
SINT16 row;
SINT16 column;

Disconnect Directory Name

SINT16fd_disconnect(fd)

FDfd;

Display Graphics Input Cursor

SINT16v_dspcur(dev_handle,x,y)

SINT16 dev_handle;
SINT16 x;
SINT16 y;

Enter Cursor Addressing Mode

```
SINT16v_enter_cur (dev_handle)  
    SINT16 dev_handle;
```

Erase To End Of Line

```
SINT16v_eeol (dev_handle)  
    SINT16 dev_handle;
```

Erase To End Of Screen

```
SINT16v_eeos (dev_handle)  
    SINT16 dev_handle;
```

Exit Cursor Addressing Mode

```
SINT16v_exit_cur (dev_handle)  
    SINT16 dev_handle;
```

File Size

```
SINT32 fd_size (fd)  
    FDfd;
```

Hardcopy

```
SINT16v_hardcopy (dev_handle)  
    SINT16 dev_handle;
```

Inquire Addressable Character Cells

SINT16vq_chcells (dev_handle,rows, columns)

SINT16 dev_handle;
*SINT16*rows;*
*SINT16*columns;*

Inquire Alpha Text Capabilities

SINT16vqa_cap (dev_handle,alph_cap)

SINT16 dev_handle;
SINT16 alph_cap[15];

Inquire Alpha Text Cell Location

SINT16vqa_cell (dev_handle,row,column, propflag,x_out,y_out)

SINT16 dev_handle;
SINT16 row;
SINT16 column;
*SINT16*propflag;*
*SINT16*x_out;*
*SINT16*y_out;*

Inquire Alpha Text Font Capability

SINT16vqa_font (dev_handle,font_in, size_in,font_status)

SINT16 dev_handle;
SINT16 font_in;
SINT16 size_in;
SINT16 font_status[7];

Inquire Alpha Text Position

SINT16 vqa_position (dev_handle,x_out, y_out)

SINT16 dev_handle;
*SINT16*x_out;*
*SINT16*y_out;*

Inquire Alpha Text String Length

SINT16 vqa_length (dev_handle,string)

SINT16 dev_handle;
 CHAR string[];

Inquire Background Mode

SINT16 vqb_mode (dev_handle)

SINT16 dev_handle;

Inquire Bitmap Formats

SINT16 vqb_format (dev_handle)

SINT16 dev_handle;

Inquire Byte Pixel Array

SINT16 vqb_pixels (dev_handle,origin,width,height,valid_width,valid_height,pixels)

SINT16 dev_handle;
 SINT16 origin[2];
 SINT16 width;
 SINT16 height;
SINT16 valid_width[2]
SINT16 valid_height[2];
 CHAR pixels[];

Inquire Cell Array

SINT16vq_cellarray (dev_handle,xy, row_length,num_rows, el_per_row,rows_used, status,colors)

```
SINT16 dev_handle;  
SINT16 xy[4];  
SINT16 row_length;  
SINT16 num_rows;  
SINT16 *el_per_row;  
SINT16 *rows_used;  
SINT16 *status;  
SINT16 colors[row_length *  
num_rows];
```

Inquire CGI Error

```
SINT16vq_error ();
```

Inquire Clip Rectangle

SINT16vqc_rectangle (dev_handle,clip_rect)

```
SINT16 dev_handle;  
SINT16 clip_rect[4];
```

Inquire Color Representation

SINT16vq_color (dev_handle,ind_in, set_flag,rgb)

```
SINT16 dev_handle;  
SINT16 ind_in;  
SINT16 set_flag;  
SINT16 rgb[3];
```

Inquire Current CursorTextAddress

SINT16vq_curaddress (dev_handle,row, column)

```
SINT16 dev_handle;  
SINT16*row;  
SINT16*column;
```

Inquire Current Fill Area Attributes

SINT16vqf_attributes (dev_handle,attrib)

```
SINT16 dev_handle;  
SINT16 attrib[4];
```

Inquire Current Graphics Text Attributes

SINT16vqt_attributes (dev_handle,attrib)

```
SINT16 dev_handle;  
SINT16 attrib[10];
```

Inquire Current Polyline Attributes

SINT16vql_attributes (dev_handle,attrib)

```
SINT16 dev_handle;  
SINT16 attrib[4];
```

Inquire Current Polymarker Attributes

SINT16vqm_attributes (dev_handle,attrib)

```
SINT16 dev_handle;  
SINT16 attrib[4];
```

Inquire Cursor Description

SINT16 vq_cursor (dev_handle, cursor_handle, data_bitmap, mask_bitmap, x, y)

```
SINT16 dev_handle;  
SINT16 cursor_handle;  
SINT16*data_bitmap;  
SINT16*mask_bitmap;  
SINT16*x;  
SINT16*y;
```

Inquire Displayable Bitmaps

SINT16 vq_displays (dev_handle, n, displays)

```
SINT16 dev_handle;  
SINT16 n;  
SINT16 displays[n];
```

Inquire Drawing Bitmap

SINT16 vqd_bitmap (dev_handle, bitmap_handle, xy)

```
SINT16 dev_handle;  
SINT16*bitmap_handle;  
SINT16 xy[4];
```

Inquire File Status

SINT16 fd_inq (fd)

```
FD fd;
```

Inquire Fill Area Representation

SINT16 vqf_representation (dev_handle, n, attrib)

```
SINT16 dev_handle;  
SINT16 n;  
SINT16 attrib[n];
```

Inquire Graphics Input Cursor

SINT16vqg_cursor (dev_handle, cursor_handle)

SINT16 dev_handle;
*SINT16*cursor_handle;*

Inquire Graphics Text Extent

SINT16vqt_extent (dev_handle,xin,yin, string,xcon,ycon,rectangle,bool)

SINT16 dev_handle;
 SINT16xin;
 SINT16yin;
 CHARstring[];
*SINT16*xcon;*
*SINT16*ycon;*
 SINT16rectangle[8];
BOOLEANbool[5];

Inquire Graphics Text Font Character

SINT16vqtf_character (dev_handle, character,n,metrics)

SINT16 dev_handle;
 SINT16 character;
 SINT16n;
SINT16metrics[n];

Inquire Graphics Text Font Description

SINT16vqtf_description (dev_handle,n, description,font_name)

SINT16 dev_handle;
 SINT16n;
SINT16description[n];
 CHARfont_name[];

Inquire Graphics Text Font Metrics

SINT16 vqtf_metrics (dev_handle, n, metrics)

SINT16 dev_handle;
SINT16 n;
SINT16 metrics[n];

Inquire Graphics Text Representation

SINT16 vqt_representation (dev_handle, n, attrib)

SINT16 dev_handle;
SINT16 n;
SINT16 attrib[20];

Inquire Input Extent

SINT16 vqi_extent (dev_handle, rectangle)

SINT16 dev_handle;
SINT16 rectangle[4];

Inquire Integer Pixel Array

SINT16 vqi_pixels (dev_handle, origin, width, height, valid_width, valid_height, pixels)

SINT16 dev_handle;
SINT16 origin[2];
SINT16 width;
SINT16 height;
SINT16 valid_width[2];
SINT16 valid_height[2];
SINT16 pixels[n];

Inquire Optimum Pattern Size

SINT16 vqo_pattern (dev_handle, xy)

SINT16 dev_handle;
SINT16 xy[2];

Inquire Polyline Representation

SINT16vql_representation (dev_handle,n, attrib)

SINT16 dev_handle;
SINT16n;
SINT16 attrib[n];

Inquire Polymarker Representation

SINT16vqm_representation (dev_handle,n, attrib)

SINT16 dev_handle;
SINT16n;
SINT16 attrib[n];

LoadCGI

SINT32 cgi_load (where,bytes_avail)

BIT8 *where;
SINT32 bytes_avail;

Message

SINT16v_message (dev_handle,message, wait)

SINT16 dev_handle;
CHAR message[];
SINT16 wait;

Open File

SINT16fd_open (fd,name,mode)

FD fd;
CHAR name[];
SINT16 mode;

Open Workstation

SINT16v_opnwk (work_in,dev_handle, work_out)

```
SINT16 work_in[19];  
SINT16 *dev_handle;  
SINT16 work_out[66];
```

Output Alpha Text

SINT16v_atext (dev_handle,string,x_out, y_out)

```
SINT16 dev_handle;  
CHAR string[ ];  
SINT16 *x_out;  
SINT16 *y_out;
```

Output Arc

SINT16v_arc (dev_handle,x,y,radius, begang, endang)

```
SINT16 dev_handle;  
SINT16 x,y;  
SINT16 radius;  
SINT16 begang;  
SINT16 endang;
```

Output Bar

SINT16v_bar (dev_handle,xy)

```
SINT16 dev_handle;  
SINT16 xy[4]
```

Output Byte Pixel Array

SINT16vb_pixels (dev_handle,origin,width, height,valid_width,valid_height, pixels)

```
SINT16 dev_handle;
SINT16 origin[2];
SINT16 width;
SINT16 height;
SINT16 valid_width[2];
SINT16 valid_height[2];
CHAR pixels[];
```

Output Cell Array

SINT16v_cellarray (dev_handle,xy, row_length,el_per_row,
num_rows,wr_mode,colors)

```
SINT16 dev_handle;
SINT16 xy[4];
SINT16 row_length;
SINT16 el_per_row;
SINT16 num_rows;
SINT16 wr_mode;
SINT16 colors[row_length*num_rows];
```

Output CGI Error

SINT16v_perror (string)

```
char string[];
```

Output Circle

SINT16v_circle (dev_handle,x,y, radius)

```
SINT16 dev_handle;
SINT16 x;
SINT16 y;
SINT16 radius;
```

Output Cursor Addressable Text

SINT16v_curtex (dev_handle, string) "

SINT16 dev_handle;
CHAR string[];

Output Ellipse

SINT16v_ellipse (dev_handle, x, y, x_radius, y_radius)

SINT16 dev_handle;
SINT16 x;
SINT16 y;
SINT16 x_radius;
SINT16 y_radius;

Output Elliptical Arc

SINT16vel_arc (dev_handle, x, y, x_radius, y_radius, s_angle, e_angle)

SINT16 dev_handle;
SINT16 x;
SINT16 y;
SINT16 x_radius;
SINT16 y_radius;
SINT16 s_angle;
SINT16 e_angle;

Output Elliptical Pie Slice

SINT16vel_pieslice (dev_handle, x, y, x_radius, y_radius, s_angle, e_angle)

SINT16 dev_handle;
SINT16 x;
SINT16 y;
SINT16 x_radius;
SINT16 y_radius;
SINT16 s_angle;
SINT16 e_angle;

Output Filled Area

SINT16v_fillarea (dev_handle,count,xy)

```
SINT16 dev_handle;  
SINT16 count;  
SINT16 xy[2*count];
```

Output Graphics Text

SINT16v_gtext (dev_handle,x,y,string)

```
SINT16 dev_handle;  
SINT16 x;  
SINT16 y;  
CHAR string[ ];
```

Output Integer Pixel Array

SINT16vi_pixels (dev_handle,origin,width,height,valid_width,valid_height, pixels)

```
SINT16 dev_handle;  
SINT16 origin[2];  
SINT16 width;  
SINT16 height;  
SINT16 valid_width[2]  
SINT16 valid_height[2];  
SINT16 pixels[ ];
```

Output Pie Slice

SINT16v_pieslice (dev_handle,x,y,radius,begang,endang)

```
SINT16 dev_handle;  
SINT16 x;  
SINT16 y;  
SINT16 radius;  
SINT16 begang;  
SINT16 endang;
```

Output Polyline

SINT16v_pline (dev_handle,count,xy)

```
SINT16 dev_handle;  
SINT16 count;  
SINT16 xy[2*count];
```

Output Polymarker

SINT16v_pmarker (dev_handle,count,xy)

```
SINT16 dev_handle;  
SINT16 count;  
SINT16 xy[2*count];
```

Parse File Name

SINT16fd_parse (qualified_name,buffer, space_available)

```
CHAR qualified_name[ ];  
CHAR buffer[ ];  
SINT16 space_available;
```

Read Cursor Keys

SINT16vrd_curkeys (dev_handle, input_mode,direction,key)

```
SINT16 dev_handle;  
SINT16 input_mode;  
SINT16 *direction;  
CHAR *key;
```

Read Directory

SINT16fd_directory (fd,name,buffer, space_available)

```
FD fd;  
CHAR name[ ];  
CHAR buffer[ ];  
SINT16 space_available;
```

Read File Proceed

```
SINT16fdp_read (fd,count,buffer)
```

```
    FDfd;  
    SINT16count;  
    CHARbuffer[];
```

Read File Wait

```
SINT16fd_read (fd,count,buffer)
```

```
    FDfd;  
    SINT16count;  
    CHARbuffer[];
```

Remove CGI

```
SINT16cgi_kill ()
```

Remove Graphics Input Cursor

```
SINT16v_rmcu (dev_handle)
```

```
    SINT16 dev_handle;
```

Rename File

```
SINT16fd_rename (fd,old_name, new_name)
```

```
    FDfd;  
    CHARold_name[];  
    CHARnew_name[];
```

Request Choice

SINT16vrq_choice (dev_handle, ch_in, ch_out)

```
SINT16 dev_handle;  
SINT16 ch_in;  
SINT16*ch_out;
```

Request Locator

SINT16vrq_locator (dev_handle, xy_in, ink, rubberband, echo_handle, xy_out, terminator)

```
SINT16 dev_handle;  
SINT16 xy_in[2];  
SINT16 ink;  
SINT16 rubberband;  
SINT16 echo_handle;  
SINT16 xy_out[2];  
CHAR *terminator;
```

Request String

SINT16vrq_string (dev_handle, max_length, echo_mode, echo_xy, string)

```
SINT16 dev_handle;  
SINT16 max_length;  
SINT16 echo_mode;  
SINT16 echo_xy[2];  
CHAR string [ ];
```

Request Valuator

SINT16vrq_valuator (dev_handle, val_in, echo_handle, val_out)

```
SINT16 dev_handle;  
SINT16 val_in;  
SINT16 echo_handle  
SINT16*val_out;
```


Reset To Defaults

```
SINT16vr_defaults (dev_handle)  
    SINT16 dev_handle;
```

Reverse Video Off

```
SINT16v_rvoff (dev_handle)  
    SINT16 dev_handle;
```

Reverse Video On

```
SINT16v_rvon (dev_handle)  
    SINT16 dev_handle;
```

Sample Choice

```
SINT16vsm_choice (dev_handle,ch_out)  
    SINT16 dev_handle;  
    SINT16*ch_out;
```

Sample Locator

```
SINT16 vsm_locator (dev_handle,xy_in, xy_out,pressed,released, key_state)  
    SINT16 dev_handle;  
    SINT16xy_in[2];  
    SINT16xy_out[2];  
    BIT16*pressed;  
    BIT16*released;  
    BIT16*key_state;
```

Sample String

INT16 vsm_string (dev_handle, max_length, echo_mode, echo_xy, string)

```
SINT16 dev_handle;  
SINT16 max_length;  
SINT16 echo_mode;  
SINT16 echo_xy[2];  
CHAR string[];
```

Sample Valuator

SINT16 vsm_valuator (dev_handle, val_out)

```
SINT16 dev_handle;  
SINT16 *val_out;
```

Seek File

SINT32 fd_seek (fd, position, whence)

```
FD fd;  
SINT32 position;  
SINT16 whence;
```

Select Drawing Bitmap

SINT16 vsd_bitmap (dev_handle, bitmap_handle)

```
SINT16 dev_handle;  
SINT16 bitmap_handle;
```

Select Graphics Input Cursor

SINT16 vsg_cursor (dev_handle, cursor_handle)

```
SINT16 dev_handle;  
SINT16 cursor_handle;
```

Set Alpha Text Color Index

SINT16 vsa_color (dev_handle, ind_in)

SINT16 dev_handle;
SINT16 ind_in;

Set Alpha Text Font And Size

SINT16 vsa_font (dev_handle, font_in, size_in, font_cap)

SINT16 dev_handle;
SINT16 font_in;
SINT16 size_in;
SINT16 font_cap[8];

Set Alpha Text Line Spacing

SINT16 vsa_spacing (dev_handle, spac_in)

SINT16 dev_handle;
SINT16 spac_in;

Set Alpha Text Overstrike Mode

SINT16 vsa_overstrike (dev_handle, mode_in)

SINT16 dev_handle;
SINT16 mode_in;

Set Alpha Text Pass Through Mode

SINT16 vsa_passthru (dev_handle, mode_in)

SINT16 dev_handle;
SINT16 mode_in;

SetAlpha TextPosition

SINT16vsa_position (dev_handle,x_in,y_in,x_out,y_out)

```
SINT16 dev_handle;  
SINT16 x_in;  
SINT16 y_in;  
SINT16 *x_out;  
SINT16 *y_out;
```

SetAlpha Text Quality

SINT16vsa_quality (dev_handle,mode_in)

```
SINT16 dev_handle;  
SINT16 mode_in;
```

SetAlpha TextSubscript/Superscript Mode

SINT16vsa_supersub (dev_handle,mode_in)

```
SINT16 dev_handle;  
SINT16 mode_in;
```

SetAlpha Text Underline Mode

SINT16vsa_underline (dev_handle, mode_in)

```
SINT16 dev_handle;  
SINT16 mode_in;
```

SetBackground Color Index

SINT16vsb_color (dev_handle,ind_in)

```
SINT16 dev_handle;  
SINT16 ind_in;
```

SetBackground Mode

SINT16vsb_mode (dev_handle,mode)

SINT16 dev_handle;
SINT16mode;

SetClip Rectangle

SINT16vsc_rectangle (dev_handle, clip_rectangle)

SINT16 dev_handle;
SINT16 clip_rectangle[4];

Set Color Representation

SINT16vs_color (dev_handle,ind_in, rgb_in,rgb_out)

SINT16 dev_handle;
SINT16 ind_in;
SINT16 rgb_in[3];
SINT16 rgb_out[3];

Set Color Table

SINT16vsc_table (dev_handle,s_index,n, rbg)

SINT16 dev_handle;
SINT16 s_index;
SINT16 n;
SINT16 rbg[n][3];

SetCursorTextAttributes

SINT16vcur_att (dev_handle,req_att, sel_att)

SINT16 dev_handle;
SINT16 req_att[4];
SINT16 sel_att[4];

Set Cursor Text Color Index

SINT16 cur_color (dev_handle, fore_requested, back_requested,
fore_selected, back_selected)

```
SINT16 dev_handle;  
SINT16 fore_requested;  
SINT16 back_requested;  
SINT16 *fore_selected;  
SINT16 *back_selected;
```

Set Deferral Mode

SINT16 vs_defer (dev_handle, mode)

```
SINT16 dev_handle;  
SINT16 mode;
```

Set Fill Area Representation

SINT16 vsf_representation (dev_handle, n, attrib)

```
SINT16 dev_handle;  
SINT16 n;  
SINT16 attrib[];
```

Set Fill Color Index

SINT16 vsf_color (dev_handle, ind_in)

```
SINT16 dev_handle;  
SINT16 ind_in;
```

Set Fill Interior Style

SINT16 vsf_interior (dev_handle, styl_in)

```
SINT16 dev_handle;  
SINT16 styl_in;
```

Set Fill Style Index

SINT16vstf_style (dev_handle,ind_in)

SINT16 dev_handle;
SINT16 ind_in;

Set Graphics Text Alignment

SINT16vst_alignment (dev_handle,hor_in,vert_in,hor_out,vert_out)

SINT16 dev_handle;
SINT16 hor_in;
SINT16 vert_in;
*SINT16*hor_out;*
*SINT16*vert_out;*

Set Graphics Text Character Height

SINT16vst_height (dev_handle,rq_height,char_width,cell_width,cell_height)

SINT16 dev_handle;
SINT16 rq_height;
*SINT16*char_width;*
*SINT16*cell_width;*
*SINT16*cell_height;*

Set Graphics Text Color Index

SINT16vst_color (dev_handle,ind_in)

SINT16 dev_handle;
SINT16 ind_in;

Set Graphics Text Font

SINT16vst_font (dev_handle,font_in)

SINT16 dev_handle;
SINT16 font_in;

Set Graphics Text Representation

SINT16 vst_representation (dev_handle, n, attrib)

SINT16 dev_handle;
SINT16 n;
SINT16 attrib[n];

Set Graphics Text String Baseline Rotation

SINT16 vst_rotation (dev_handle, ang_in)

SINT16 dev_handle;
SINT16 ang_in;

Set Input Extent

SINT16 vsi_extent (dev_handle, rectangle)

SINT16 dev_handle;
SINT16 rectangle[4];

Set Line Edit Characters

SINT16 vs_editchars (dev_handle, line_del, char_del)

SINT16 dev_handle;
CHAR line_del;
CHAR char_del;

Set Pen Speed

SINT16 vs_penspeed (dev_handle, speed)

SINT16 dev_handle;
SINT16 speed;

SetPolyline Color Index

SINT16 vsl_color (dev_handle, ind_in)

SINT16 dev_handle;
SINT16 ind_in;

Set Line Cross Section

SINT16 vsl_cross (dev_handle, bitmap_handle)

SINT16 dev_handle;
SINT16 bitmap_handle;

SetLine Type

SINT16 vsl_type (dev_handle, type_in)

SINT16 dev_handle;
SINT16 type_in;

SetLine Width

SINT16 vsl_width (dev_handle, wid_in)

SINT16 dev_handle;
SINT16 wid_in;

SetPolyline Representation

SINT16 vsl_representation (dev_handle, n, attrib)

SINT16 dev_handle;
SINT16 n;
SINT16 attrib[n];

SetPolymarker ColorIndex

SINT16 vsm_color (dev_handle, ind_in)

SINT16 dev_handle;
SINT16 ind_in;

SetPolymarker Height

SINT16 vsm_height (dev_handle, hgt_in)

SINT16 dev_handle;
SINT16 hgt_in;

SetPolymarker Representation

SINT16 vsm_representation (dev_handle, n, attrib)

SINT16 dev_handle;
SINT16 n;
SINT16 attrib[n];

SetPolymarker Type

SINT16 vsm_type (dev_handle, type_in)

SINT16 dev_handle;
SINT16 type_in;

SetUserLine Type

SINT16 vsul_type (dev_handle, pattern)

SINT16 dev_handle;
SINT32 pattern;

Set Writing Mode

SINT16vswr_mode (dev_handle,mode_in)

SINT16 dev_handle;
SINT16 mode_in;

Update Workstation

SINT16v_updwk (dev_handle)

SINT16 dev_handle;

Write File Proceed

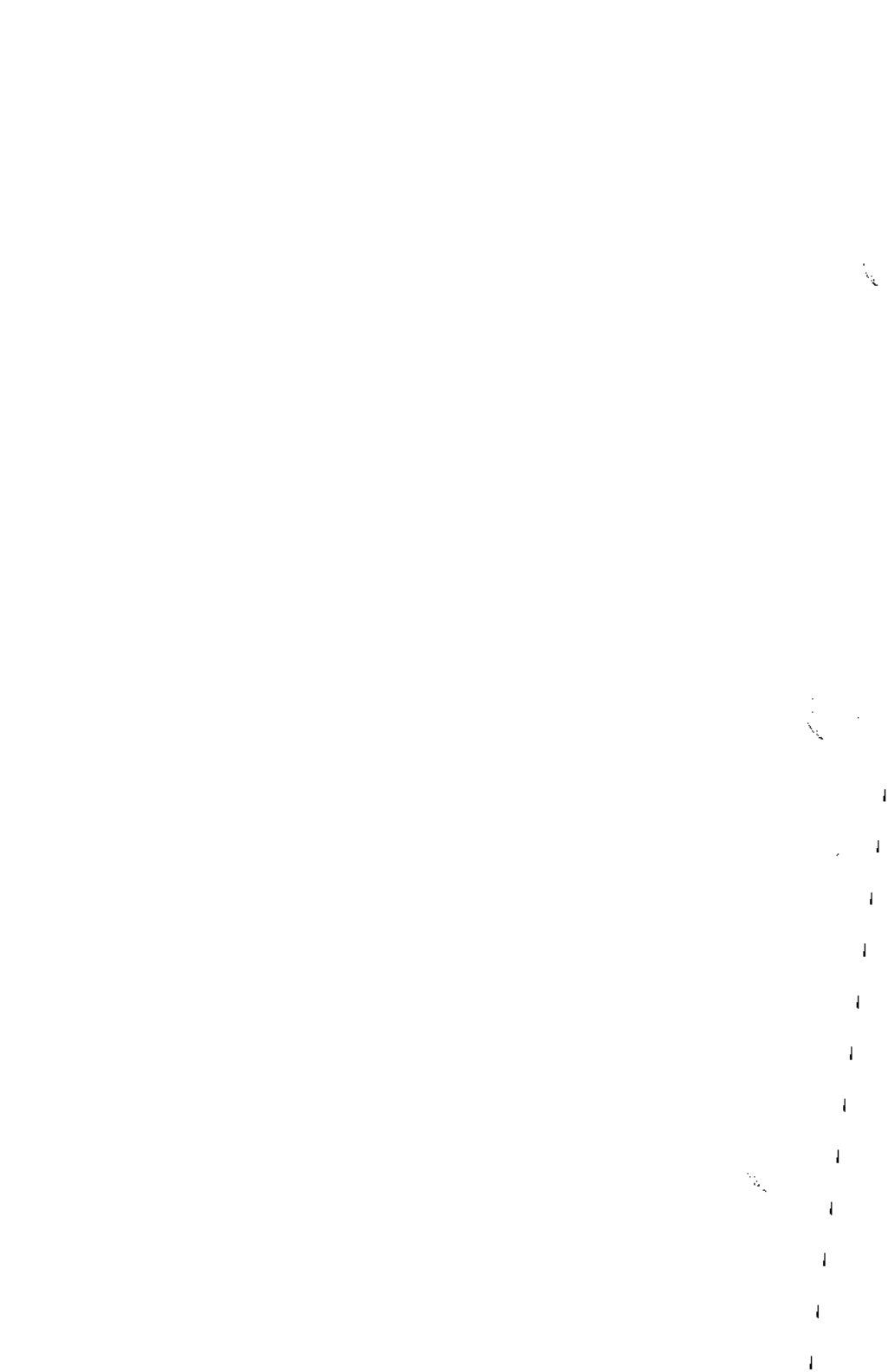
SINT16fdp_write (fd,buffer,count)

FD fd;
CHAR buffer[];
SINT16 count;

Write File Wait

SINT16fd_write (fd,buffer,count)

FD fd;
CHAR buffer[];
SINT16 count;



C

1

Q

6-12-87
514-210-040