Silicon Graphics, Inc.

IRIS

IRIS Terminal Programming Environment

Silicon Graphics, Inc.
630 Clyde Court
Mtn. View, CA 94043

Release Staff:

Glen Williams
Donl Mathis
Bob Toxen

IRIS Terminal Programming Environment [GL2]
Document number: 5001-022-001-0

CONTENTS

1. The IRIS Terminal Programming Environment

This document describes the construction of a custom IRIS terminal program and its corresponding remote graphics library. The programming environment is supported on the IRIS workstation and builds programs that run on the IRIS terminal.

With the programming environment, you can add or delete routines from the standard graphics library. Added routines allow you to run complex, interactive code segments on the terminal giving faster response (mainly because network and remote host delays are eliminated). These added routines can be called from the host or from within the terminal program. Unused routines can be removed from the standard library to save memory space on the terminal. Using these techniques, you can write terminal programs that run totally within the terminal, as some of the SGI demonstration programs do.

1.1 Setting Up Your Environment Variables

The terminal programming environment is set up to reside anywhere in the UNIX file system. All the makefiles refer to their targets relative to some environment variables. Before running any of the makefiles, you must set up your environment properly.

Make a directory to hold your terminal programming environment. For the purposes of this document, assume that you called it /usr/progenv. Cd to that directory, and read the contents of the tape to it. (If you ordered this software with your machine, look on your disk in directory /usr/src/tpe for the code). You should have the following subdirectories:

doc	**dllib**	**engr**
host	**internet**	**term**

To set up your environment correctly, issue the following commands:

```
setenv BASE /usr/progenv/engr
setenv DESTDIR /usr/progenv/engr
setenv MACHINE MC68000
setenv SYSTEM SYSTEM5
```

You also need to ensure that your BOOT environment variable is set to an appropriate directory. Inside that directory, create a sub-directory called test.

The makefiles put files into $BOOT/test. When you are satisfied they work, move them up to $BOOT.

Next, source the file **env.csh** (if you are using csh — the C shell) or **env.sh** (if you use sh — the Bourne shell). After modifying the file **lib.prim** in **/usr-/progenv/engr/iris/lib** as described below, cd to **/usr/progenv/host/c/src/gl** and do a

```
make install
```

After modifying the terminal program, connect to **/usr/progenv/engr/term** and do a

```
make install
```

The make command will make two terminal programs: — iris and tcpiris. Iris is for serial, XNS connections or IEEE 488 connections and tcpiris is for serial or tcp connections. To save time, you can issue a command for a specific connection; for example, use

```
make iris
```

1:2 How the IRIS Terminal Program Works

The IRIS terminal program consists of three parts: a communication section, a terminal emulator, and a dispatch routine.

The **communication section** controls the network connection (where "network" is taken to mean RS232, ethernet (XNS or IP/TCP), or IEEE 488 — any reliable byte-stream protocol). The computer on the other end of this network is called the **remote host** and programs that run there control the IRIS terminal program.

The **terminal emulator** behaves like a standard ASCII terminal. Characters sent to this routine are drawn on the textport and certain escape characters have special interpretations (insert line, move cursor, clear textport, etc.).

The **dispatch routine** reads characters from the network, performs the appropriate function on the IRIS, and perhaps returns characters to the remote host. When graphics programs are not being run on the remote host, this usually consists of sending every character to the terminal emulator part of the program. If the IRIS is not in graphics mode, the dispatch routine also sends keystrokes from the keyboard to the remote host. If the graphical escape character is sent by the remote host, the dispatch routine will go into graphics mode and will interpret the next few characters as a graphics command.

The graphical part of the dispatch routine is completely table-driven. The format of the table below is artificially simple — the exact details appear later in this document — but it shows how the dispatch routine works.

Dispatch Commands		
Token	*Command*	*Parameters*
1	move	"fff"
2	move2i	"ll"
3	clear	""
4	color	"s"
5	isobj	"lB"
6	getmatrix	"F:16"

All the routines in the graphics library (as well as a few others) appear in this table. The characters in the "parameters" column indicate the types of arguments the commands take. "fff" means that the move command takes three floating-point numbers; "ll" means two longs (32-bit integers); "" means that the clear command has no parameters; "s" means a short (a 16-bit integer); "lB" means that the isobj command is sent a long and returns a byte (boolean values are returned as bytes). The last example, getmatrix, requires no input parameters and returns 16 floating-point numbers.

Every graphics command from the remote host is preceded by a graphics escape character, indicated here by <GESC>[1]. The command token is sent encoded as two bytes and is followed by byte-encoded versions of all the other input parameters. The dispatch routine decodes the command and its parameters and then calls the command on the IRIS. If any values are returned, they are sent by the dispatch routine to the remote host.

1.3 Adding Commands to the Graphics Library

To add a new command to the graphics library, you must:

1. Write and test (on the workstation, if possible), the routine to be added.
2. Make an appropriate entry into the command table.
3. Run a makefile that automatically generates a new version of the IRIS terminal program and of the remote graphics library stubs.

1. <GESC> is currently ASCII 16, or control-P, but may change in the future.

As an example, suppose you wish to make a graphics library command that clears the screen to a given color, and then returns the number of bitplanes on the system. Write and debug the following routine on the workstation until you are certain it does what you want:

```
short funnycolor(col)
Colorindex col;
{
     color(col);
     clear();
     return getplanes();
}
```

If you invoke a line of code that reads

```
x = funnycolor(BLACK);
```

on the remote host, you would like the screen to be cleared to black and the number of available bitplanes on the IRIS to be returned in x.

The makefiles are set up so that simplest change — adding a single routine to the remote graphics library — requires changing only two files. These are $IRIS/lib/lib.prim and $IRIS/src/term/local.c. Local.c should contain the source code for all the additional routines, and lib.prim describes the parameters and return values of each of the graphics library routines.

Lib.prim is used by awk scripts to generate both the dispatch table in the IRIS terminal program and the remote host stubs for C and FORTRAN. The first part of the file contains documentation for the table entries and serves as a good source of examples. Most routines can be added to the list by following the pattern of a similar existing entry. Details of the lib.prim entry format are provided in the next section.

The position of an entry in the lib.prim file determines its dispatch number. New routines should therefore be added to the end of the list; otherwise all previously compiled programs may become incompatible. Additions or deletions from the middle of the list will cause this problem. To delete a routine from the standard library, replace the entry with:

```
V:V:bogus( )
```

Note that the list already contains some "bogus" entries. They hold places for commands from older versions of the graphics library that have changed or disappeared. At the end of the list is another special entry called "lastone". New entries to the list should be made just before "lastone."

The makefile in the $IRIS/src/term directory assumes that all additional source code for the terminal program appears in the file local.c. If the terminal additions are extensive, you can add any number of files to that makefile.

1.4 Lib.prim Entries

This section describes in detail the entries in the lib.prim file. Most routines can be added by modeling your new entry after one that is already in the list. For most routines, a complete understanding of this section is unnecessary. It is much easier to follow the discussion below if you have a copy of the lib.prim file in front of you.

Each entry in the lib.prim file has the following general form:

```
        <return type>:<procedure name>( <parameter description> )
```

The <procedure name> is the name of the routine, and should be unique in the first six characters. The <parameter description> is a list of entries from the table of defined types in the comment at the beginning of the lib.prim file. The <return type> is slightly different and will be described later.

Each entry in the <parameter description> is either a pair or a triplet separated by colons. The first letter in each pair or triplet describes the type that will be generated by the awk script. In the C version, for example, "a" generates type "char", "k" generates type "Colorindex", and so on.

The second part of each entry (also a letter) is somewhat redundant information that tells the physical type of the entry. This could be looked up in a table, but is included so that the awk scripts will run faster. Lower-case letters are used if the parameter is sent by the remote host; upper-case is used for parameters received. For example, in the entry "k:s", the "k" means that the logical type is "Colorindex" and the "s" means that a Colorindex is actually a 16-bit short. Since "s" is lower-case, this means that a short is transmitted from the host to the terminal.

The third part of triplet entries is used for lengths of arrays of items. It can be a constant or can have the form arg<m> or <n>*arg<m>, where <n> and <m> are constants (arg5 or 3*arg4, for example). If it is a constant, then it is the absolute size of the array. Arrays whose size depends on other parameters to the function are described with the other form. For example, the actual entry for poly is:

```
        V:V:poly( u:l L:f:3*arg1 )
```

The first entry is "u:l", meaning that the first parameter from the poly routine is of type integer and is transmitted as a 32-bit long. The second entry, "L:f:3*arg1", means that the next parameter is of type "Coord _[][]", the data to be transmitted is of type float, and the number of floats to be transmitted is three times the value of the first argument to the routine.

Some of the entries in the table have the following form: "I:f:len,F:len,F". This means that any of these forms are legal: "I:f:len", "I:F:len", "I:F".

<return type> is similar to the entries in the <parameter description>. Note that entries with a non-void <return type> always return values to the host — so in all cases, the second part of the entry is in upper-case. To return a short, use the entry "e:S". The "e" tells the awk script to use the type "short" and that the value is sent as a 16-bit short. It might seem that "f:S" should be used as listed in the defined types in the comment in lib.prim, but this would cause the awk scripts to generate:

```
short *foo();
```

instead of:

```
short foo();
```

Basically, the problem is the difference between variables appearing on the left- and right-hand side of an assignment. The assignment "a = b" takes the *value* of b and puts it into the *location* of a.

1.5 IRIS Terminal Routines

There are some restrictions on the routines that run on the IRIS.

Your new routines can call any commands in the graphics library and can contain arbitrary stand-alone code. UNIX Software does not run on the terminal, however, so do not make any UNIX system calls. In particular, there is no file system. If you need to transfer files to and from the IRIS terminal, they must be sent and returned as arrays.

In general, do not use any of the I/O routines from the standard C library, or you will interfere with the operation of the terminal program. To do I/O, use graphics library commands (getvaluator(), getbutton(), etc.).

Caution: the libraries that you load *do* contain many of the standard C I/O routines. The terminal program uses them for its I/O (to deal with the physical keyboard, etc.). If you call them, you will get unpredictable (and possibly catastrophic) results. Since you are building the whole IRIS program, these routines must appear in the library. It might be possible eventually to re-arrange the libraries in such a way that there is less danger, but this has not been done so far.

You can use the standard storage management routines — malloc(), free(), and friends — and the math routines. In fact, routines that are usually considered part of the I/O library and that are not related to physical I/O can also be used. Sprintf(), for example, can be used.

All the routines in the graphics library that are not exported have names beginning with "gl_". If you avoid these names, names in the terminal program itself, and the C I/O routines mentioned above, you should not have any problems with name conflicts.

Some other cautions are in order, and although they may seem obvious, they are worth stating. The list below is not complete, but gives a flavor of the dangers:

1. Infinite loops will cause the terminal program to hang, since the dispatch routine just waits for the local routine to return before it continues with the next command.
2. Your new routine runs in the same address space as the rest of the graphics library, and there is no array-bounds checking, etc. Bad code can write over the entire terminal program. Malloc() and free() use the same free list as the rest of the graphics library, so errors in storage allocation can crash the terminal.
3. If you call makeobj() but not closeobj() and then call a custom routine that calls makeobj(), you will get the same result as you would by calling makeobj() twice in a row from the remote host — i.e., probably not exactly what you want.

Silicon Graphics, Inc.

630 Clyde Court
Mountain View
California 94043