GetStatus

```
C:    int _GetStatus(vga_status_rec *stat_rec)
```

Size: $01 bytes

$00: $00 Command Code

Return data:
(the following data is stored starting at the pointer stat_rec)

$00: 'G' Ascii values of the letters, used as an identification
$01: 'S' that a Second Sight card is in fact installed in this machine
$02: 'V'
$03: 'G'
$04: 'A'
$05: size_rec
  Number of bytes of record data that follow
$06: Second Sight Firmware Version Number
  bits 7-4: major revision number
  bits 3-0: minor revision number
  (Example: version 1.0 is $10, version 4.6 = $46)
$07: emul_status
  0 = emulating, !0 = not emulating
$08: vga_mode
  Mode number of last set video mode
$09: video_ram
  0 = 512K, 1 = 1MB
$0A: monitor_type
  IS_VGA (0x10) VGA monitor
  0x00 AppleColor RGB monitor

Looks for a Second Sight video card in the current system. If no card is found, this routine returns a -1.

SetMode

```
C:    void _SetMode(int modeNum, int emulFlag)
```

Size: $03 bytes

$00: $01 Command Code
$01: $XX Screen Mode to Switch To
  (VGA BIOS Screen Mode Parameter)

$01 - 40x25
$03 - 80x25
$53 - 640x480x256 (VGA only)
$13 - 320x200x256
$61 - 640x400x256
$FA - 560x192x16
$FB - 280x192x16
$FC - 40x24
$FD - 80x24
$FE - 640x200

$02: $EE Emulation Flag
$01 = Do not emulate the current Apple II video mode
$00 = Emulate current Apple II video mode

Upload Code/Data

C: `_UploadData(int flag, void *dst, longword len, void *src)`

Size: $0B bytes

$00: $02 Command Code
$01: $XX Code/Data Flag
$02: $0ABBCC Address in Z180 memory to put data
$05: $0LLLLL Length of data block to transfer
$08: $AABBCC Address in IIGS memory to take data from

Flags: `FL_CODE = 0, FL_DATA = 1`

Transfers a block of data from the IIGS memory address src, and length len, to the Second Sight card. If the "Code/Data" flag is FL_DATA, the `dst` address refers to an offset in the VGA controller's video memory. If the "Code/Data" flag is FL_CODE, the `dst` is a direct Z180 address.

Caution!: Due to the fact that the 1MB of video memory is bank switched in two 512K pieces during thhis operation (at address 0x080000), you cannot transfer data across the 512K boundary. If you do transfer data across this boundary, the Z180's DMA controller will wrap around to address zero, overwriting the firmware, and you will crash the Second Sight (and thus the IIGS).

For example, if you wanted to transfer 768K of data from the IIGS to address 0x010000 in the VGA memory, you would need to do two transfers:

0x070000 bytes at address 0x010000
0x050000 bytes at address 0x080000

Second Sight will see that the first transfer starts at less than bank 08, and will select the low VGA memory bank. On the second transfer, it will see that the

bank is 08 or above and select the high VGA memory bank. Second Sight cannot switch banks in the middle of a transfer, thus the need for you to break up the transfer.

(In the future, the `_UploadData` command may handle this automatically, in which case the two separate transfers would still be compatible, but just an extra step).

The C library routine `vgaUploadVideoData` automatically handles this necessity; see the library docs for details.

Upload Bitmap (Not currently implemented)

Size: $08 bytes

$00: $0	
$01: $LLLL	Number of lines in the bitmap
$03: $WWWW	Width (# bytes wide screen is, minus # bytes pixmap)
$05: $0AAAAAA	Starting address in VGA memory (vga memory only)

Scroll Screen

C: `void _ScrollScreen(void *src, void *dst, longword length)`

Size: $0A bytes

$00: $03	Command Code
$01: $0SSSSS	Offset in VGA memory to begin move at
$04: $0DDDDD	Destination of copy command
$07: $0LLLLL	Number of bytes to move

Scrolls the entire contents of the screen in a direction specified by the two parameters. The caller is responsible for clearing the sections of the screen that should be cleared.

Effectively, to scroll the screen by pixel values, the final offset to use in the memory copy command is:

v * num_pixels_per_line + h

This routine only works in the lower 512K of video memory, as of ROM version 1.1. There is currently no way to move data above the 512K boundary.

Screen Off

C: `void _ScreenOff(void)`

Size: $01 bytes

$00: $04 Command Code

Disables screen output. This can be useful for those situations where you want to "smoothly" change the screen mode, or fill the screen with data, and then show it all at once. Also, since screen output is disabled, accesses to the video memory will not have to wait for video refresh cycles, speeding up throughput somewhat (at a 1MB/sec maximum DMA transfer speed, this shouldn't really affect throughput much, if at all).

Screen On

C: `void _ScreenOn(void)`

Size: $01 bytes

$00: $05 Command Code

Reenables screen output.

SetPalette

C: `void _SetPalette(rgb_24 *palette)`

Size: $04 bytes

$00: $06	Command Code
$01: $AABBCC	Address in IIGS memory of palette data 256 entries of 3 bytes each, or 768 bytes

Uploads a complete new palette to the VGA controller. The palette consists of 256 palette entries, each three bytes. The bytes are:

$00: Red
$01: Green
$02: Blue

The RGB values have a full 8-bit width, so there are a maximum of 16 million colors to select from.

SetPaletteEntry

C: `_SetPaletteEntry(int entry, rgb_32 triplet)`

Size: $05 bytes

$00: $07	Command Code
$01: $PN	Palette entry number to change
$02: $AABBCC	One RGB triplet

Modifies a single VGA palette entry in the VGA controller. The palette entry consists of three bytes:

$02: Red
$03: Green
$04: Blue

The RGB values each have a full 8-bit width, so there are a maximum of 16.7 million colors to select from.

SetBorder

C: `_SetBorder(int entry)`

Size: $02 bytes

$00: $08	Command Code
$01: $XX	Palette entry color used to describe the border color

---

Run Code

Size: $06 bytes

$00: $09	Command Code
$01: $AA	Value of CBR register
$02: $BB	Value of CBAR register
$03: $CC	Value of BBR register
$04: $AABB	Address in Z180 memory to CALL to

CBR, CBAR, and BBR must be carefully set to avoid crashing the Second Sight board. The lower nibble of CBAR **MUST** be 2 (indicating that the first bank-switched area starts at $2000). This is because the Second Sight's command interpreter and interrupt firmware reside from $0000 - $1FFF in the address space. Once your code is running, you can change this if you disable interrupts and are careful to restore state before executing the 'RET' instruction.

As of ROM 1.1, this routine does appear to work.

Clear Screen

C: `void _ClearScreen(int color, void *dst, longword len)`

Size: $08

$00: $0A	Command Code
$01: $CC	Color to set the pixels to
$02: $0ABBCC	Address to start setting pixels at
$05: $0LLLLL	Number of bytes to set to that value

Sets the range of video memory starting at address `dst` and of length `len` to `color` (a byte value).

This routine only works in the lower 512K of video memory.

SetShadow

C: `void _SetShadow(int flag)`

Size: $02

$00: $0B	Command Code
$01: $XX	0 = GS Video shadowing will occur 4 = do not shadow GS video data writes

---

SetVGAReg

C: `void _SetVGAReg(int idx, int idxval, int reg, int val)`

An index/register pair is specified, as well as a value to store into that VGA register.

Size: $07

$00: $0C	Command Code
$01: $IL	low byte of index register
$02: $IH	high byte of index register
$03: $II	Index register value
$04: $RL	Low byte of register address
$05: $RH	high byte of register address
$06: $VV	Value to store into register

GetVGAReg

C: `int _GetVGAReg(int idx, int idxval, int reg)`

An index/register pair is specified, and the value of the register is returned.

Size: $06

$00: $0D	Command Code
$01: $IL	low byte of index register
$02: $IH	high byte of index register
$03: $II	Index register value
$04: $RL	Low byte of register address
$05: $RH	high byte of register address

Return value:

$00: $VV	Value returned from register

SetUserMode

C: `void _SetUserMode(vga_mode_rec *table)`

Size: $01

$00: $0E Command number

Followed by 84 bytes of mode selection data (see **vga_mode_rec**).

WARNING: Some cheap VGA monitors can actually blow up their circuit boards if you pass bogus data to this routine and then call _SetMode(0xFF,1). **BE SURE YOU KNOW WHAT YOU ARE DOING!! IF YOU DON'T KNOW WHAT YOU'RE DOING, STICK WITH THE PROVIDED VIDEO MODES. Sequential Systems will accept no responsibility for monitors damaged in this way.**

Multi-Sync monitors and the AppleColor RGB monitor are robust enough to handle garbage mode settings appropriately (the AppleColor will shut down, MultiSyncs will try to sync and if they fail will do nothing).

This routine may, in the future, checksum the mode data as an aid to preventing accidentally passing bad data through should the machine crash or what-not.

SetTextFont

Size: $04

$00: $0F	
$01: $XX	Font number to select

Sets the current text font used by the Second Sight. The values for XX are as follows:

$00:	Standard ROM font
$01:	Alternate ROM font
$02:	Standard PC ANSI Font

$03: User Uploaded Font (must be at address $00F000 in the SRAM)

If a text mode is currently active, the font change takes place immediately.

This function is not implemented as of ROM 1.1.

Z180 Memory Map

The following two tables diagram the Second Sight's memory map as seen by the Z180 processor.

$000000	
.	
.	128K SRAM - IIGS video shadowing, program storage (see diagram below)
.	
$01FFFF	
$040000	
.	
.	a softswitch - ***do not modify***!!!
.	
$05FFFF	
$060000	
.	
.	128K EPROM -
.	
$07FFFF	
$080000	
.	
.	VGA Video Memory
.	
$0FFFFF	

SRAM Map

Bank 0

$0000 - $03FF	misc. code, interrupt vectors, stack
$0400 - $0BFF	Text pages 1 and 2 shadow buffers
$0C00 - $1FFF	Command handler firmware, interrupt handlers
$2000 - $5FFF	Hires Page 1 and Page 2 shadow buffers
$6000	Misc. Shadowing code
$C000 - $CFFF	Video mode table storage
$FFFF	

Bank 1

$0000	free space
$2000 - $9FFF	Super Hires/Double Hires Shadow buffers

$A000 - $BFFF	SHR Shadowing code (only when SHR Shadowing is active and running)
$C000	Free Space
$FFFF	

If you can guarantee that a particular IIGS video page will not be written to by your program, you can put code there. Otherwise, to use the video page areas for program storage you must make the `_SetShadow` call and disable video shadowing (Second Sight V1.1 EPROM **and** Lattice or greater). With shadowing inactive, you may use much of Bank 0 (both hires pages, and $D000-$FFFF) and all of Bank 1.

The SHR Shadowing code is copied to its running place when emulation is enabled and SHR mode is turned on. When SHR mode is off, or emulation is disabled, the area from $A000 - $BFFF in bank 1 is unused.