# **Permuted Index**

0 0 1 32 bit D/VRAM copy 32-bit copy with variable	PMv0put(3X) PMv(3X)
1	PMv(3X)
<ol> <li>1</li></ol>	• •
32 bit D/VRAM copy	DMT 2 V
32-bit copy with variable	
32-bit DRAM or VRAM copy	
3D vector and return its length	
access video and Z memory /to	
address to a section of DRAM	
allocate a DRAM block	PMgetzdesc(3X)
allocation /PMset_hireg reserve	
alternate pipes of a dual pipe/	
angle PMcos trigonometric function	
apply a function to all subscreens	PMapply(3X)
array of longs to float	
assembler	d3as(1)
available to a user program	DEVopen(3S)
begin execution of all pipe and	DEVrun(3H)
begin execution of programs loaded	DEVpipe run(3S)
begin execution of programs loaded	
bit D/VRAM copy	
block PMgetzdesc,	PMgetzdesc(3X)
block PMinterleave	
block of data to a pipe DSP's PDR	DEVpipe put(3S)
block of data to a pixel DSP's PDR	
block of four byte values from a	
block of four byte values to a pipe	
block of memory from a pipe DSP	DEVpipe read(3S)
block of memory from a pixel DSP	DEVpixel read(3S)
block to a reserved location in a/	
blocks of VRAM	PMcopyvtov(3X)
board DEVfifo reset	
board and return the value /update	
board and returns value /read the	
board mode register	
board to operate in parallel mode	DEVfifo parallel(3S)
board to operate in serial mode	
broadcast bus is granted	
buffer DEVget_pixel, DEVget_pixels	
buffer DEVget scan line read	
buffer DEVput_pixel, DEVput_pixels	
buffer PMgetpix	
buffer PMgetzbuf	
buffer PMputpix	
buffer PMputzbuf	
buffer PMqget quick	
buffer PMqput quick	
buffer	
buffer	
buffer 0	

PMv0get read a pixel from bu PMv0put write a pixel to but PMv1get read a pixel from but PMv1put write a pixel to but PMcopy f fast but danger increments PMcopy PMcopy\_s s PMnorm normaliz manipulate page registers used /load a page register and return PMgetzdesc, PMzdesc va DRAM and page registers for dynamic DEVswap pipe switch primary to compute the cosine of PMap PMlong\_dsp convert d3as DS /DEVopen\_system make a Pixel mach pixel nodes DEV into specified pipe/ DEVpipe into specified pixel/ DEVpixel PMcopy f fast but dangerous PMzdesc valid allocate a DR. interleave or deinterleave register DEVpipe\_put wri register DEVpixel put ser pipe feedback/ DEVfifo read rea FIFO DEVfifo write wri DEVpipe\_read read DEVpixel read rea DEVpixel id write write a nod PMcopyvtov c resets all FIFOs on a color tables from video contro color tables from video contro DEVpixel\_mode\_init initialize p DEVfifo parallel configure a DEVfifo serial configure a /wait until control of read a pixel from the fra one or more scan lines from a fra write pixels into the fra read a pixel from the cur read a float value from th output a pixel to the cur write a float value to th read of a pixel from the cur write of a pixel to the cur PMswapback swap meaning of b PMzget read a float from the PMv0get read a pixel f PMv0put write a pixe

buffer 1 ..... PMv(3X) buffer 1 ..... PMv(3X) buffer of bytes from the Z memory ..... DEVread z(3S) buffer of bytes into the Z memory ...... DEVwrite z(3S) buffer to a pipe DSP ..... DEVpipe write(3S) buffer to a pixel DSP ..... DEVpixel\_write(3S) buffer to be displayed ...... DEVpixel buffer(3S) buffering mode ...... PMdblbuff(3X) buffering mode ...... PMsnglbuff(3X) buffers PMswapbuff ...... PMswapbuff(3X) bus is granted PMswap pipe ..... PMswap pipe(3H) byte values from a pipe feedback/ ..... DEVfifo read(3S) byte values to a pipe FIFO ...... DEVfifo write(3S) bytes from a pixel DSP's PIR/ ..... DEVpixel\_get(3S) bytes from the PIR of a pipe DSP ..... DEVpipe get(3S) bytes from the Z memory of a pixel ..... DEVread z(3S) bytes into the Z memory of a pixel ..... DEVwrite z(3S) C compiler for Pixel Machine ...... devcc(1) calibration values and sets color ...... DEVload color tables(3S) call DEVexit /wait for pixel ..... DEVwait\_exit(3H) called /define a message ...... DEVuser msg enable(3H) check status of node's ID ..... DEVpipe id check(3S) check status of node's ID ..... DEVpixel id check(3S) clear ..... PMwaitsem(3N) clear the software semaphore in the ..... DEVrelease\_pipe\_semaphore(3H) code and specify functions to be/ ..... DEVuser msg enable(3H) code that uses the print routines ...... devprint(1) color lookup tables /reads file of ..... DEVload color tables(3S) color lookup tables from shadow/ ...... DEVshadow\_off(3S) color lookup tables from shadow/ ..... DEVshadow on(3S) color tables from video controller ...... DEVput color map(3S) color tables from video controller ...... DEVget color map(3S) color value /macro that converts ..... PMfloat color(3N) color value PMint color macro that ..... PMint color(3N) color value to an integer ..... PMcolor\_int(3N) color value to floating point/ ..... PMcolor float(3N) column PMzaddrcol ..... PMzaddrcol(3X) command from a pixel node FIFO ..... PMgetcmd(3X) commands PMcommand ...... PMcommand(4N) commands PMcommand ..... PMcommand(4N) commands PMenable enable ..... PMenable(3N) commands back from the feedback/ ..... DEVwrite(3H) commands to the regular output FIFO ..... PMfb on(3P) compiler ..... d3cc(1) compiler for Pixel Machine programs ...... devcc(1) completion of a Pixel Machine/ ..... PMhost exit(3N) completion, then call DEVexit ...... DEVwait exit(3H) compute the cosine of an angle ..... PMcos(3M) configure a pipe board to operate ...... DEVfifo\_parallel(3S)

**DEVtools Reference Manual** 

of a pixel node DEVread z read a of a pixel/ DEVwrite z writes a DEVpipe write write a DEVpixel write write a DEVpixel buffer selects the frame PMdblbuff enable double PMsnglbuff disable double swap front and back pixel wait until control of the broadcast DEVfifo read read a block of four DEVfifo write write a block of four DEVpixel\_get read a stream of DEVpipe\_get read a stream of node DEVread z read a buffer of node DEVwrite z writes a buffer of programs using DEVtools devcc d3cc DSP32 lookup tables /reads file of gamma nodes to signal completion, then code and specify functions to be DEVpipe id check DEVpixel id check PMwaitsem wait for semaphore to memory/ /DEVrelease\_pixel\_semaphore DEVexit halts processors, **DEV**close DEVuser msg enable define a message /server program for Pixel Machine gamma calibration values and sets DEVshadow off turns off updating of DEVshadow on turns on updating of board and/ DEVput color map update board/ DEVget color map read the floating point value to internal converts an integer to an internal /macro that converts internal /macro that converts internal generate a ZRAM pointer to a PMgetcmd load data structure used for FIFO data structure used for FIFO processing of selected system /Pixel Machines pipelines and read PMfb on direct output d3cc DSP32 C language using DEVtools devcc C

PMv1get read a pixel from

PMv1put write a pixel to

using DEVtools devcc C /to the host that signals the /wait for pixel nodes to signal PMcos trigonometric function to in parallel mode DEVfifo parallel

in serial mode DEVfifo_serial	configure a pipe board to operate	DEVfifo_serial(3S)
granted PMswap_pipe wait until	control of the broadcast bus is	PMswap_pipe(3H)
/update color tables from video	controller board and return the/	DEVput_color_map(3S)
/read the color tables from video	controller board and returns value	DEVget_color_map(3S)
printf formatted output	conversion on host	printf(3N)
PMlong_dsp	convert an array of longs to float	PMlong_dsp(3M)
and host long integer DEVbswapl	convert between DSP32 long integer	DEVbswapl(3S)
and host long integer DEVsswapl	convert between DSP32 long integer	- · ·
and host short integer DEVbswaps	convert between DSP32 short integer	
host long integers DEVswap_long	convert from DSP32 long integers to	
to host short/ DEVswap_short	convert from DSP32 short integers	
oating-point format/ DEVdsp_ieee	convert from the DSP32	
ating-point format/ DEVieee_dsp	convert from the host's	
PMieee_dsp	convert IEEE float to DSP float	PMieee_dsp(3M)
olor value PMint_color macro that	converts an integer to an internal	
internal/ PMfloat_color macro that	converts floating point value to	
integer PMcolor_int macro that	converts internal color value to an	PMcolor_int(3N)
floating/ PMcolor_float macro that	converts internal color value to	
PMmyx test if a given screen space	coordinate is in processor space	PMmyx(3X)
PMmyy test if a given screen space	coordinate is in processor space	PMmyy(3X)
PMxat map subscreen	coordinates to screen space	PMxat(3X)
PMyat map subscreen	coordinates to screen space	PMyat(3X)
fast but dangerous 32 bit D/VRAM	copy PMcopy_f	PMcopy_f(3X)
by_s safe 32-bit DRAM or VRAM	сору	PMcopy_s(3X)
PMcopyvtov	copy blocks of VRAM	PMcopyvtov(3X)
	copy DRAM to video RAM	(3X)
another PMcopyztoz	copy from one section of DRAM to	
another PMqcopyztoz	copy from one section of DRAM to	PMqcopyztoz(3X)
PMcopyftob	copy front to back	PMcopyftob(3X)
data from input to/ PMcopycmd	copy opcode, parameter count, and	
PMcopyvtoz	copy video RAM to DRAM	
PMcopy_v 32-bit	copy with variable increments	
function to compute the	cosine of an angle /trigonometric	
Mcopycmd copy opcode, parameter	count, and data from input to/	
PMputcmd write opcode, parameter	count, and parameters to the output/	
PMgetop get opcode and parameter	count from input FIFO of a pipe/	
Aputop write opcode and parameter	count to the output FIFO of a pipe/	
PMgetpix read a pixel from the	current buffer	
PMputpix output a pixel to the	current buffer	
quick read of a pixel from the	current buffer PMqget	
quick write of a pixel to the	current buffer PMqput	
	d3as DSP32 assembler	
	d3cc DSP32 C language compiler	
	d3ld DSP32 link editor	
	d3sim DSP32 link editor	
PMcopy_f fast but	dangerous 32 bit D/VRAM copy	PMcopy_f(3X)
PMgetdata get	data from a pipe node FIFO	PMgetdata(3P)
/copy opcode, parameter count, and	data from input to output FIFO of a/	
PMmsg_exchange send and receive	data packet over serial links	
commands PMcommand	data structure used for FIFO	
commands PMcommand	data structure used for FIFO	
DEVpipe_put write a block of	data to a pipe DSP's PDR register	DEVpipe_put(3S)

/update color ta /read the color ta printf and host long integ and host long inter and host short integ host long integers to host short/ floating-point format floating-point format color value PMint\_ internal/ PMfloat integer PMcolo floating/ PMcolor PMmyx test if a give PMmyy test if a gi PMxa PMya fast but dangerous PMcopy\_s safe 32-bit D anoth

anothe data from input P function PMcopycmd copy or PMputcmd write of PMgetop get opcod PMputop write opcod PMgetpix read PMputpix outp quick read of quick write

PMcopy_f fast bu
PMgetdata ge
/copy opcode, parameter count, and
PMmsg_exchange send and receive
commands PMcommand
commands PMcommand
DEVpipe_put write a block of

data to a pixel DSP's PDR register ..... DEVpixel put(3S) decrement references to a page ...... PMfreezaddr(3X) define a message code and specify ..... DEVuser msg enable(3H) DEVbswapl convert between DSP32 ..... DEVbswapl(3S) DEVbswaps convert between DSP32 ..... DEVbswaps(3S) devcc C compiler for Pixel Machine ..... devcc(1) DEVclose closes the Pixel Machine ...... DEVclose(3S) DEVcread, DEVreadn, DEVreadn alt, ..... DEVwrite(3H) DEVcwrite, DEVwriten, DEVcwriten, ..... DEVwrite(3H) DEVcwriten, DEVwrite\_alt, DEVcread / ..... DEVwrite(3H) devdisp download an image from a ..... devdisp(1) DEVdsp ieee convert from the DSP32 ..... DEVdsp ieee(3S) DEVerror generate an error message ..... DEVerror(3S) DEVexit /wait for pixel nodes ..... DEVwait exit(3H) DEVexit halts processors, closes ..... DEVexit(3H) DEVfifo parallel configure a pipe ...... DEVfifo parallel(3S) DEVfifo read read a block of four ...... DEVfifo read(3S) DEVfifo reset resets all FIFOs on a ..... DEVfifo reset(3S) DEVfifo\_serial configure a pipe ...... DEVfifo serial(3S) DEVfifo write write a block of four ..... DEVfifo write(3S) DEVget color map read the color ...... DEVget color map(3S) DEVget image header read the Pixel ..... DEVget image header(3S) DEVget pixel, DEVget pixels read a ..... DEVget pixel(3S) DEVget\_pixels read a pixel from the ..... DEVget\_pixel(3S) DEVget scan line read one or more ..... DEVget scan line(3H) device DEVexit halts ..... DEVexit(3H) device DEVinit ..... DEVinit(3H) DEVieee dsp convert from the host's ..... DEVieee dsp(3S) DEVimage\_header format of a ..... DEVimage header(4) DEVimage header format of a ...... DEVimage header(4) DEVinit opens and initializes Pixel ...... DEVinit(3H) DEVlast\_pipe, DEVpixel\_nodes,/ ..... DEVpixel\_system(3S) DEVlast\_pixel, DEVx\_nodes,/ ..... DEVpixel\_system(3S) DEVload color tables reads file of ..... DEVload color tables(3S) DEVlock manage Pixel Machine locks ...... DEVlock(3S) DEVmodel code, DEVvideo code, ..... DEVpixel system(3S) DEVopen, DEVopen system make a ...... DEVopen(3S) DEVopen\_system make a Pixel machine ...... DEVopen(3S) DEVpipe\_boot load a Pixel Machine ..... DEVpipe\_boot(3S) DEVpipe get read a stream of bytes ..... DEVpipe get(3S) DEVpipe get msg read a message from ...... DEVpipe get msg(3S) DEVpipe get pir read the PIR ..... DEVpipe get pir(3S) DEVpipe\_halt halt a pipe node ...... DEVpipe\_halt(3S) DEVpipe id check check status of ...... DEVpipe id check(3S) DEVpipe id print read and print the ..... DEVpipe id print(3S) DEVpipe\_nodes, DEVlast\_pipe, ..... DEVpixel\_system(3S) DEVpipe\_put write a block of data ..... DEVpipe\_put(3S) DEVpipe read reads a block of ...... DEVpipe read(3S) DEVpipe run begin execution of ...... DEVpipe run(3S) DEVpipe\_write write a buffer to a ..... DEVpipe\_write(3S) DEVpixel\_boot load a Pixel Machine ...... DEVpixel\_boot(3S)

DEVpixel\_put send a block of register PMfreezaddr functions to be/ DEVuser\_msg\_enable PMinterleave interleave or long integer and host long integer short integer and host short/ programs using DEVtools

macros/ /DEVcwriten, DEVwrite alt, DEVwrite alt, DEVcread,/ DEVwrite, DEVwrite, DEVcwrite, DEVwriten, file to a Pixel Machine floating-point format to the IEEE/ on standard error to signal completion, then call Pixel Machine device board to operate in parallel mode byte values from a pipe feedback/ pipe board board to operate in serial mode byte values to a pipe FIFO tables from video controller board/ Machine image header from a file pixel from the frame buffer frame buffer DEVget pixel. scan lines from a frame buffer processors, closes Pixel Machine opens and initializes Pixel Machine floating-point format to the/ DEVtools image file DEVtools image file Machine device DEVpixel system DEVpipe nodes, /DEVlast pipe, DEVpixel nodes, gamma calibration values and sets/

/DEVx\_screen, DEVy\_screen, Pixel machine available to a user/ available to a user/ DEVopen, executable into specified set of/ from the PIR of a pipe DSP the PIR of a pipe DSP processor node's ID node ID of a processor DEVpixel\_nodes,/ DEVpixel\_system to a pipe DSP's PDR register memory from a pipe DSP programs loaded into specified/ pipe DSP

executable into specified set of/

**DEVtools Reference Manual** 

DEVpixel buffer selects the frame ...... DEVpixel buffer(3S) DEVpixel\_get read a stream of bytes ...... DEVpixel get(3S) DEVpixel\_get\_msg read a message ..... DEVpixel get msg(3S) DEVpixel halt halt a pixel node ...... DEVpixel halt(3S) DEVpixel id check check status of ...... DEVpixel id check(3S) DEVpixel id print read and print ...... DEVpixel\_id print(3S) DEVpixel\_id\_write write a node id ...... DEVpixel\_id\_write(3S) DEVpixel mode init initialize pixel ...... DEVpixel mode init(3S) DEVpixel mode overlay set overlay ..... DEVpixel mode overlay(3S) DEVpixel nodes, DEVlast pixel/ ..... DEVpixel system(3S) DEVpixel\_overlay update overlay ...... DEVpixel\_overlay(3S) DEVpixel put send a block of data ...... DEVpixel put(3S) DEVpixel read read a block of ...... DEVpixel read(3S) DEVpixel run begin execution of ...... DEVpixel run(3S) DEVpixel system DEVpipe nodes, ..... DEVpixel system(3S) DEVpixel write write a buffer to a ...... DEVpixel write(3S) DEVpoll nodes poll DSP processors ...... DEVpoll nodes(3H) devprint a host server program for ...... devprint(1) DEVput color map update color ...... DEVput color map(3S) DEVput image header write a Pixel ..... DEVput image header(3S) DEVput pixel, DEVput pixels write ...... DEVput pixel(3S) DEVput scan line download an image ..... DEVput\_scan line(3H) DEVreadn, DEVreadn alt, macros to/ ..... DEVwrite(3H) DEVreadn alt, macros to write to/ ..... DEVwrite(3H) DEVread z read a buffer of bytes ..... DEVread z(3S) DEVrelease\_pipe\_semaphore, ..... DEVrelease\_pipe\_semaphore(3H) DEVrelease\_pixel\_semaphore clear ..... DEVrelease\_pipe\_semaphore(3H) DEVrun begin execution of all pipe ..... DEVrun(3H) devsave upload an image from a ...... devsave(1) DEVserial direction updates the ..... DEVserial direction(3S) DEVshadow\_off turns off updating of ..... DEVshadow\_off(3S) DEVshadow\_on turns on updating of ..... DEVshadow\_on(3S) DEVsswapl convert between DSP32 ..... DEVsswapl(3S) DEVswap long convert from DSP32 ..... DEVswap long(3S) DEVswap pipe switch primary and ..... DEVswap pipe(3H) DEVswap short convert from DSP32 ..... DEVswap short(3S) DEVtools devcc C compiler ...... devcc(1) DEVtools image file ..... DEVimage header(4) DEVtools image file ...... DEVimage header(4) DEVuser\_msg\_enable define a message ..... DEVuser\_msg\_enable(3H) DEVvideo code, /DEVx screen, ..... DEVpixel system(3S) DEVwait exit wait for pixel nodes ...... DEVwait exit(3H) DEVwrite, DEVcwrite, DEVwriten, ..... DEVwrite(3H) DEVwrite\_alt, DEVcread, DEVreadn,/ ..... DEVwrite(3H) DEVwriten, DEVcwriten, ..... DEVwrite(3H) DEVx nodes, DEVy nodes, DEVx scale,/ ..... DEVpixel system(3S) DEVx\_scale, DEVy\_scale,/ ..... DEVpixel\_system(3S) DEVx\_screen, DEVy\_screen,/ ..... DEVpixel\_system(3S) DEVy nodes, DEVx\_scale, DEVy scale,/ ..... DEVpixel system(3S)

buffer to be displayed from a pixel DSP's PIR register from a pixel DSP's PIR register register of a pixel DSP processor node's ID the node ID of a processor block to a reserved location in a/ board mode register mode in the pixel mode register /DEVpipe nodes, DEVlast pipe, mode in all pixel processor's flag/ to a pixel DSP's PDR register memory from a pixel DSP programs loaded into specified/ DEVlast\_pipe, DEVpixel\_nodes,/ pixel DSP for messages Pixel Machine code that uses the/ tables from video controller board/ Machine image header to a file pixels into the frame buffer frame buffer DEVput pixel, or a portion of an image to a/ /DEVcwriten, DEVwrite\_alt, DEVcread, /DEVwrite alt, DEVcread, DEVreadn, from the Z memory of a pixel node DEVrelease pixel semaphore clear/ the/ DEVrelease\_pipe\_semaphore, and pixel nodes Pixel Machine to a file serial I/O link direction color lookup tables from shadow/ color lookup tables from shadow/ long integer and host long integer long integers to host long/ alternate pipes of a dual pipe/ short integers to host short/ for Pixel Machine programs using DEVimage\_header format of a DEVimage header format of a code and specify functions to be/ DEVy screen, DEVmodel code, to signal completion, then call/ DEVcwriten, DEVwrite alt./ /DEVcwrite, DEVwriten, DEVcwriten, DEVwrite alt / DEVwrite, DEVcwrite, into the Z memory of a pixel node /DEVpixel nodes, DEVlast pixel, /DEVx nodes, DEVy nodes, /DEVy nodes, DEVx scale, DEVy scale, /DEVlast pixel, DEVx nodes,

DEVy scale, DEVx screen/ ..... DEVpixel system(3S) DEVy screen, DEVmodel code,/ ..... DEVpixel system(3S) direct output commands to the ..... PMfb\_on(3P) direction DEVserial direction ...... DEVserial direction(3S) direction PMsiodir(3X) disable double buffering mode ..... PMsnglbuff(3X) displayed DEVpixel buffer ..... DEVpixel buffer(3S) division ..... PMfdiv(3M) dot product for light sources ...... PMldot(3M) double buffering mode ...... PMdblbuff(3X) double buffering mode ...... PMsnglbuff(3X) download an image from a file to a ..... devdisp(1) download an image or a portion of ...... DEVput scan line(3H) DRAM ..... PMcopyvtoz(3X) DRAM /load a page register and ..... PMgetzaddr(3X) DRAM and page registers for dynamic/ ..... PMzbrk(3X) DRAM block PMgetzdesc, PMgetzdesc(3X) DRAM or VRAM copy ..... PMcopy s(3X) DRAM to another ..... PMcopyztoz(3X) DRAM to another PMqcopyztoz ..... PMqcopyztoz(3X) DSP DEVpipe get read a stream ...... DEVpipe get(3S) DSP DEVpipe get msg read ..... DEVpipe get msg(3S) DSP DEVpipe\_get\_pir ..... DEVpipe\_get\_pir(3S) DSP DEVpipe read ...... DEVpipe read(3S) DSP DEVpipe write ...... DEVpipe write(3S) DSP DEVpixel\_get\_pir ..... DEVpixel\_get\_pir(3S) DSP DEVpixel read ...... DEVpixel read(3S) DSP DEVpixel write ...... DEVpixel write(3S) DSP float ...... PMieee dsp(3M) DSP processors /software semaphore ...... DEVrelease\_pipe\_semaphore(3H) DSP processors for messages ..... DEVpoll nodes(3H) DSP32 C language compiler ...... d3cc(1) DSP32 floating point format /host's ..... DEViece\_dsp(3S) DSP32 floating-point format to the ..... DEVdsp\_ieee(3S) DSP32 long integer and host long ...... DEVbswapl(3S) DSP32 long integer and host long ...... DEVsswapl(3S) DSP32 long integers to host long ...... DEVswap long(3S) DSP32 short integer and host short ..... DEVbswaps(3S) DSP32 short integers to host short/ ..... DEVswap\_short(3S) DSP's memory /a node id block to ..... DEVpixel id write(3S) DSP's PDR register DEVpipe put ..... DEVpipe put(3S) DSP's PDR register DEVpixel put ..... DEVpixel put(3S) DSP's PIR register DEVpixel\_get ..... DEVpixel\_get(3S) DSP's PIR register ...... DEVpixel\_get msg(3S) dual pipe system /switch ..... DEVswap pipe(3H) D/VRAM copy ..... PMcopy f(3X) dynamic allocation /PMset\_hireg ..... PMzbrk(3X)

**DEVtools Reference Manual** 

updates the serial I/O link PMsiodir set serial I/O link PMsnglbuff selects the frame buffer to be PMfdiv perform floating point PMmsg\_setup set serial PMIdot specialized PMdblbuff enable PMsnglbuff disable Pixel Machine devdisp an image to a/ DEVput scan line PMcopyvtoz copy video RAM to return an address to a section of /PMset lowreg, PMset hireg reserve PMzdesc valid allocate a PMcopy s safe 32-bit PMcopyztoz copy from one section of copy from one section of copy of bytes from the PIR of a pipe a message from the PIR of a pipe read the PIR register of a pipe reads a block of memory from a pipe write a buffer to a pipe read the PIR register of a pixel read a block of memory from a pixel write a buffer to a pixel PMieee dsp convert IEEE float to in the memory of one of the DEVpoll nodes poll d3as d3cc floating-point format to the IEEE/ DEVdsp\_ieee convert from the d3ld d3sim integer DEVbswapl convert between integer DEVsswapl convert between integers DEVswap long convert from integer DEVbswaps convert between DEVswap\_short convert from a reserved location in a pixel node write a block of data to a pipe

/DEVx nodes, DEVy nodes, DEVx scale,

/DEVy scale, DEVx screen,

regular output FIFO PMfb on

send a block of data to a pipe send a block of data to a pixel read a stream of bytes from a pixel /read a message from a pixel primary and alternate pipes of a PMcopy\_f fast but dangerous 32 bit reserve DRAM and page registers for

editor	
editor	
enable double buffering mode	PMdblbuff(3X)
enable processing of selected	PMenable(3N)
error DEVerror generate	DEVerror(3S)
error message on standard error	DEVerror(3S)
executable into specified set of/	
executable into specified set of/	
execution of all pipe and pixel	DEVnum(3H)
execution of programs loaded into	
execution of programs loaded into	DEVnixel mn(35)
fast but dangerous 32 bit D/VRAM	
Tast but dangerous 52 bit D/VRAW	
feedback FIFO /read a block	DEVIIIO_read(35)
feedback FIFO /Machines pipelines	DEVwrite(3H)
PM_FLAG LED on or off	
PM_RDY LED on or off	PMrdyled(3X)
FIFO /read a block of four	DEVfifo_read(3S)
FIFO DEVfifo_write write a	DEVfifo_write(3S)
FIFO /Machines pipelines and read	DEVwrite(3H)
FIFO PMfb_on direct output	PMfb on(3P)
FIFO PMgetcmd	PMgetcmd(3X
FIFO	PMgetdata(3P
FIFO commands	PMcommand(4N
FIFO commands	
FIFO of a pipe node /count,	
FIFO of a pipe node /get opcode	
FIFO of a pipe node /parameter	DMputemd(2D
FIFO of a pipe node PMputdata	DMputdate(0D
FIFO of a pipe node /write opcode	PMpuldala(3P
FIFO of a pipe node /write opcode	
FIFOs on a pipe board	DE Vfilto_reset(3S
file DEVget_image_header read the	DEVget_image_header(3S
file DEVimage_header	
file DEVimage_header	DEVimage_header(4
file DEVput_image_header write	DEVput_image_header(3S
file devsave upload	
file of gamma calibration values	
file to a Pixel Machine	devdisp(1
fill a rectangular region of the	PMclear(3X
flag registers /update overlay	
float PMieee dsp	PMieee dsb(3M
float PMlong_dsp	PMlong_dsp(3M
float from the z buffer	PMznet/3X
float to DSP float	PMiego dsp/3M
float to the Z-buffer	
fleat value from the 7 buffer	
float value from the Z buffer	PMgetzbuf(3X
float value to the Z buffer	PMputzbut(3X
floating point division	
floating point format /the host's	
floating point number /macro that	PMcolor_float(3N
floating point value to internal/	PMfloat_color(3N
floating-point format /the DSP32	DEVdsp_ieee(3S
floating-point format to the DSP32/	DEVieee dsp(3S

d3ld DSP32 link d3sim DSP32 link PMdblbuff system commands PMenable an error message on standard DEVerror generate an DEVpipe boot load a Pixel Machine DEVpixel boot load a Pixel Machine nodes DEVrun begin specified pipe/ DEVpipe\_run begin specified pixel/ DEVpixel\_run begin copy PMcopy f of four byte values from a pipe and read commands back from the PMflagled turn the PMrdyled turn the byte values from a pipe feedback block of four byte values to a pipe commands back from the feedback commands to the regular output load command from a pixel node PMgetdata get data from a pipe node PMcommand data structure used for PMcommand data structure used for and data from input to output and parameter count from input count, and parameters to the output write parameters to the output and parameter count to the output DEVfifo\_reset resets all Pixel Machine image header from a format of a DEVtools image format of a DEVtools image a Pixel Machine image header to a an image from a Pixel Machine to a and/ DEVload color tables reads devdisp download an image from a screen PMclear mode in all pixel processor's convert IEEE float to DSP convert an array of longs to PMzget read a PMieee\_dsp convert IEEE PMzput write a PMgetzbuf read a PMputzbuf write a PMfdiv perform floating-point format to the DSP32 converts internal color value to PMfloat color macro that converts floating-point format to the IEEE DEVieee dsp convert from the host's

floating-point format to the IEEE/ ..... DEVdsp ieee(3S) format /the DSP32 floating-point ..... DEVdsp ieee(3S) format /the host's floating-point ..... DEVieee\_dsp(3S) format of a DEVtools image file ..... DEVimage\_header(4) format of a DEVtools image file ..... DEVimage header(4) format to the DSP32 floating point/ ..... DEVieee dsp(3S) format to the IEEE floating-point/ ..... DEVdsp ieee(3S) formatted output conversion on host ..... printf(3N) four byte values from a pipe/ ..... DEVfifo\_read(3S) four byte values to a pipe FIFO ...... DEVfifo write(3S) frame buffer DEVget pixel, ..... DEVget pixel(3S) frame buffer DEVget\_scan\_line ..... DEVget\_scan\_line(3H) frame buffer DEVput pixel, ..... DEVput\_pixel(3S) frame buffer to be displayed ...... DEVpixel buffer(3S) front and back pixel buffers ..... PMswapbuff(3X) front to back ..... PMcopyftob(3X) function ..... PMpow(3M) function ...... PMsin(3M) function ...... PMx exp n(3M) function of x and y from screen ..... PMfxytoij(3X) function of x from screen space to ..... PMfxtoi(3X) function of y from screen space to ..... PMfytoj(3X) function to all subscreens ..... PMapply(3X) function to compute the cosine of ..... PMcos(3M) functions to be called ..... DEVuser msg enable(3H) gamma calibration values and sets/ ..... DEVload\_color\_tables(3S) generate a pointer to a specific ..... PMpixaddr(3X) generate a ZRAM pointer to a column ...... PMzaddrcol(3X) generate a ZRAM pointer to a row ...... PMzaddr(3X) generate an error message on ...... DEVerror(3S) given screen space coordinate is in ...... PMmyx(3X) given screen space coordinate is in ..... PMmyy(3X) granted PMswap\_pipe wait until ..... PMswap\_pipe(3H) halt a pipe node processor ..... DEVpipe halt(3S) halt a pixel node processor ...... DEVpixel halt(3S) halts processors, closes Pixel ..... DEVexit(3H) header from a file ...... DEVget image header(3S) header to a file ..... DEVput image header(3S) host PMusermsg ..... PMusermsg(3N) host printf ..... printf(3N) host long integer /convert ..... DEVbswapl(3S) host long integer /convert ..... DEVsswapl(3S) host long integers DEVswap long ..... DEVswap long(3S) host server program for Pixel ..... devprint(1) host short integer /convert ..... DEVbswaps(3S) host short integers /convert ..... DEVswap\_short(3S) host that signals the completion of/ ..... PMhost exit(3N) host's floating-point format to ..... DEVieee dsp(3S) ID DEVpipe id check ..... DEVpipe id check(3S) ID DEVpixel id check ...... DEVpixel id check(3S) id block to a reserved location in ..... DEVpixel id write(3S)

DEVimage header /from the host's floating-point /from the DSP32 floating-point printf DEVfifo read read a block of DEVfifo write write a block of DEVget pixels read a pixel from the read one or more scan lines from a DEVput pixels write pixels into the DEVpixel buffer selects the PMswapbuff swap PMcopyftob copy PMpow power PMsin trigonometric PMsort soare root PMx exp n integer power space to/ PMfxytoij map a linear processor/ PMfxtoi map a linear processor/ PMfytoj map a linear PMapply apply a an angle PMcos trigonometric /define a message code and specify DEVload\_color\_tables reads file of pixel PMpixaddr **PMzaddrcol** PMzaddr standard error DEVerror processor space PMmyx test if a processor space PMmyy test if a control of the broadcast bus is **DEVpipe** halt **DEVpixel** halt Machine device DEVexit /read the Pixel Machine image /write a Pixel Machine image send a user message to the formatted output conversion on between DSP32 long integer and between DSP32 long integer and convert from DSP32 long integers to Machine code that uses/ devprint a

DEVdsp ieee convert from the DSP32

format to the IEEE floating-point format to the DSP32 floating point

**DEVimage** header

between DSP32 short integer and from DSP32 short integers to PMhost\_exit send a message to the the/ DEVieee\_dsp convert from the check status of node's check status of node's

a/ DEVpixel\_id\_write write a node

**DEVtools Reference Manual** 

ID of a processor DEVpipe_id_print ID of a processor IEEE float to DSP float IEEE floating-point format /the (ihi) PMihi map from screen (ilo) PMilo map from screen image file DEVimage_header image file DEVimage_header image from a file to a Pixel image from a Pixel Machine to a	DEVpixel_id_print(3S) PMieee_dsp(3M) DEVdsp_ieee(3S) PMihi(3X) PMilo(3X) DEVimage_header(4) DEVimage_header(4) devdisp(1)
image header from a file	DEVget image header(3S)
image header to a file	DEVput_image_header(3S)
image or a portion of an image to a	DEVput_scan_line(3H)
image to a Pixel Machine /download	DEVput_scan_line(3H)
increments	PMcopy_v(3X)
initialize pixel board mode	
initialize serial I/O	PMsioinit(3X)
initializes Pixel Machine device	DEVinit(3H)
input FIFO of a pipe node PMgetop	PMgetop(3P)
input pointer	PMmsg_setup(3X)
input to output FIFO of a pipe node	
integer DEVbswapl convert between	DEVbswapl(3S)
integer DEVbswaps convert between	
integer DEVsswapl convert between	
integer PMcolor_int macro that	
integer and host long integer	
integer and host long integer	DEVsswapl(3S)
integer and host short integerinteger power function	DEVDSWAPS(3S)
integer to an internal color value integers DEVswap_long convert from	DEVewap long(38)
integers /convert from	DEVswap_long(33)
integers to host long integers	
integers to host short integers	DEVswap_long(00)
interleave or deinterleave a block	PMinterleave(3X)
internal color value /macro that	PMfloat color(3N)
internal color value /macro	PMint_color(3N)
internal color value to an integer	
internal color value to floating/	
I/O	PMsioinit(3X)
I/O link direction	DEVserial direction(3S)
I/O link direction	PMsiodir(3X)
j /function of x and y from screen	PMfxytoij(3X)
j /map a linear function of y from	PMfytoj(3X)
(jhi) PMjhi map from screen	PMjhi(3X)
(jlo) PMjlo map from screen	PMjlo(3X)
language compiler	d3cc(1)
LED on or off	
LED on or off	
length PMnorm normalize	PMnorm(3M)
light sources	PMldot(3M)
linear function of x and y from	PMfxytoij(3X)

read and print the node /read and print the node PMieee dsp convert DSP32 floating-point format to the space (xmax) to processor space space (xmin) to processor space format of a DEVtools format of a DEVtools Machine devdisp download an file devsave upload an /read the Pixel Machine /write a Pixel Machine Pixel/ DEVput scan line download an an image or a portion of an PMcopy v 32-bit copy with variable register DEVpixel\_mode\_init PMsioinit DEVinit opens and get opcode and parameter count from PMmsg setup set serial DMA /parameter count, and data from DSP32 long integer and host long DSP32 short integer and host short DSP32 long integer and host long converts internal color value to an /convert between DSP32 long /convert between DSP32 long /convert between DSP32 short PMx\_exp\_n PMint color macro that converts an DSP32 long integers to host long DSP32 short integers to host short /convert from DSP32 long /convert from DSP32 short **PMinterleave** converts floating point value to that converts an integer to an PMcolor int macro that converts PMcolor float macro that converts PMsioinit initialize serial /updates the serial PMsiodir set serial space to processor space i and screen space to processor space space (ymax) to processor space space (ymin) to processor space d3cc DSP32 C PMflagled turn the PM FLAG PMrdyled turn the PM RDY a 3D vector and return its

PMIdot specialized dot product for screen space to/ PMfxytoij map a

linear function of x from screen	
linear function of y from screen	
lines from a frame buffer	DEVget_scan_line(3H)
link direction DEVserial_direction	DEVserial_direction(3S)
link direction	
link editor	d3ld(1)
link editor	d3sim(1)
links PMmsg exchange send	PMmsg exchange(3X)
load a page register and return an	
load a Pixel Machine executable	
load a Pixel Machine executable	
load command from a pixel node FIFO	PMgetcmd(3X)
loaded into specified pipe nodes	
loaded into specified pixel nodes	
location in a pixel node DSP's/	
location in a pixer node DSP s/	DEvpixer_id_write(33)
locks	
longs to float	
lookup tables /reads file of gamma	
lookup tables from shadow tables	
lookup tables from shadow tables	
Machine	
Machine devdisp download	devdisp(1)
Machine /download an image	
machine available to a user program	DEVopen(3S)
Machine code that uses the print/	devprint(1)
Machine device DEVexit	DEVexit(3H)
Machine device	
Machine executable into specified	
Machine executable into specified	
Machine image header from a file	DEVget image header(3S)
Machine image header to a file	
Machine locks	DEVlock(3S)
Machine program /to the host that	
Machine programs using DEVtools	
Machine to a file devsave	
Machines pipelines and read/	
macro that converts an integer to	
macro that converts floating point	
macro that converts internal color	
macro that converts internal color	PMcolor_float(3N)
macros to manipulate page registers	PMpagereg(3X)
macros to write to the Pixel/	
manage Pixel Machine locks	
manipulate page registers used to	PMpagereg(3X)
map a linear function of x and y	PMfxytoij(3X)
map a linear function of x from	PMfxtoi(3X)
map a linear function of y from	
map from screen space (xmax) to	
map from screen space (xmin) to	
map from screen space (ymax) to	
map from screen space (ymin) to	
map subscreen coordinates to screen	

**DEVtools Reference Manual** 

/read one or more scan updates the serial I/O PMsiodir set serial I/O d3ld DSP32 d3sim DSP32 and receive data packet over serial address to a section of/ PMgetzaddr into specified set of/ DEVpipe boot into specified set/ DEVpixel boot PMgetcmd /begin execution of programs /begin execution of programs /write a node id block to a reserved **DEVlock manage Pixel Machine** PMlong dsp convert an array of calibration values and sets color /turns off updating of color /turns on updating of color **DEVclose closes the Pixel** an image from a file to a Pixel or a portion of an image to a Pixel /DEVopen\_system make a Pixel /a host server program for Pixel halts processors, closes Pixel **DEVinit opens and initializes Pixel** set of/ DEVpipe boot load a Pixel set of/ DEVpixel boot load a Pixel DEVget image header read the Pixel DEVput image header write a Pixel DEVlock manage Pixel signals the completion of a Pixel devcc C compiler for Pixel upload an image from a Pixel /macros to write to the Pixel an internal color/ PMint color value to internal/ PMfloat color value to an integer PMcolor int value to floating/ PMcolor float used to/ PMpagereg, PMdesc, PMxlate /DEVcread, DEVreadn, DEVreadn alt, DEVlock access/ /PMdesc, PMxlate macros to from screen space to/ PMfxytoij screen space to processor/ PMfxtoi screen space to processor/ PMfytoj processor space (ihi) PMihi processor space (ilo) PMilo

space to processor/ PMfxtoi map a space to processor/ PMfytoj map a

processor space (ilo) PMilo processor space (ilo) PMilo processor space (ilo) PMilo processor space (ilo) PMilo space PMxat

at	map subscreen coordinates to screen	PMyat(3X)
ар	meaning of back buffer	PMswapback(3)
's	memory /node id block to a reserved	
z	memory /manipulate page registers	
of	memory from a pipe DSP	DEVpipe read(3S)
of	memory from a pixel DSP	DEVoixel read(3S)
Z	memory of a pixel node DEVread z	
z	memory of a pixel node DEVwrite z	
ne	memory of one of the DSP processors	D211110_1(00)
	DEV	(release nine semanhore(3H)
el	memory without subscreens /or write	PMgetrow(3X)
a	message code and specify functions	
a a	message from a pixel DSP's PIR	
a a	message from the PIR of a pipe DSP	
a or	message on standard error	DEVpror(3S)
	message to the host	DEVENDI(33)
er a	message to the host that signals	
or	messages DEVpoll_nodes	
	messages DEvpoinnodes	
lel	mode DEVfifo_parallel configure	
ial	mode DEVfifo_serial configure	
ng	mode	
ng	mode	
ay	mode in all pixel processor's flag/	DEVpixel_overlay(3S)
ay	mode in the pixel mode register	DEVpixel_mode_overlay(3S)
rd	mode register DEVpixel_mode_init	DEVpixel_mode_init(3S)
el	mode register	
el	node DEVread_z read a buffer of	
el	node DEVwrite_z writes a buffer of	
ре	node /parameter count, and data	
ре	node /get opcode and parameter	
pe	node /count, and parameters	
ре	node PMputdata write parameters	
ре	node /write opcode and parameter	
(el	node DSP's memory /a node id block	DEVpixel_id_write(3S)
kel	node FIFO	PMgetcmd(3X)
ре	node FIFO	
a	node id block to a reserved	DEVpixel_id_write(3S)
he	node ID of a processor	DEVpipe_id_print(3S)
he	node ID of a processor	DEVpixel_id_print(3S)
ре	node processor	
kel	node processor	DEVpixel_halt(3S)
ре	nodes /a Pixel Machine executable	DEVpipe boot(3S)
ре	nodes /begin execution of	DEVpipe run(3S)
kel	nodes /a Pixel Machine executable	DEVpixel boot(3S)
kel	nodes /begin execution of programs	DEVpixel run(3S)
kel	nodes DEVrun begin	DEVrun(3H)
of	node's ID	
of	node's ID	DEVpixel id check(3S)
kel	nodes to signal completion, then	DEVwait exit(3H)
rm	normalize a 3D vector and return	
do	nothing for a specified time	PMdelav(3N)
int	number /that converts internal	PMcolor float(3N)

space PMyat PMswapback swap location in a pixel node DSP's used to access video and Z DEVpipe\_read reads a block of DEVpixel\_read read a block of read a buffer of bytes from the Z writes a buffer of bytes into the Z /clear the software semaphore in the

a scanline or scancolumn from pixe to be/ DEVuser msg enable define register DEVpixel get msg read DEVpipe\_get\_msg read DEVerror generate an erro PMusermsg send a use the completion/ PMhost exit send poll DSP processors for a pipe board to operate in paralle a pipe board to operate in seria PMdblbuff enable double bufferin PMsnglbuff disable double bufferin DEVpixel\_overlay update overla DEVpixel mode overlay set overla initialize pixel boar /set overlay mode in the pixe bytes from the Z memory of a pixe bytes into the Z memory of a pixe from input to output FIFO of a pip count from input FIFO of a pip to the output FIFO of a pip to the output FIFO of a pip count to the output FIFO of a pip to a reserved location in a pix PMgetcmd load command from a pixe PMgetdata get data from a pip location/ DEVpixel id write write DEVpipe\_id\_print read and print th /read and print th DEVpipe halt halt a pip DEVpixel halt halt a pix into specified set of pip programs loaded into specified pip into specified set of pixe loaded into specified pix execution of all pipe and pix DEVpipe id check check status DEVpixel id check check status call/ DEVwait exit wait for pix its length PMnor PMdelay d color value to floating poi

opcode and parameter count from ..... PMgetop(3P) opcode and parameter count to the ..... PMputop(3P) opcode, parameter count, and data ..... PMcopycmd(3P) opcode, parameter count, and ..... PMputcmd(3P) opens and initializes Pixel Machine ..... DEVinit(3H) operate in parallel mode ..... DEVfifo parallel(3S) operate in serial mode ..... DEVfifo\_serial(3S) output a pixel to the current ...... PMputpix(3X) output a value to the PIR register ...... PMoutpir(3N) output commands to the regular ..... PMfb on(3P) output conversion on host ...... printf(3N) output FIFO PMfb on direct ..... PMfb on(3P) output FIFO of a pipe node ..... PMcopycmd(3P) output FIFO of a pipe node ..... PMputcmd(3P) output FIFO of a pipe node ..... PMputdata(3P) output FIFO of a pipe node /write ..... PMputop(3P) overlay mode in all pixel/ ..... DEVpixel overlay(3S) overlay mode in the pixel mode ..... DEVpixel mode overlay(3S) overlay on or off ..... PMoverlay(3P) packet over serial links ..... PMmsg exchange(3X) page register PMfreezaddr ..... PMfreezaddr(3X) page register and return an address ...... PMgetzaddr(3X) page registers for dynamic/ ..... PMzbrk(3X) page registers used to access video ..... PMpagereg(3X) parallel mode /configure ...... DEVfifo parallel(3S) parameter count, and data from ...... PMcopycmd(3P) parameter count, and parameters to ..... PMputcmd(3P) parameter count from input FIFO of ..... PMgetop(3P) parameter count to the output FIFO ...... PMputop(3P) parameters to the output FIFO of a/ ..... PMputcmd(3P) parameters to the output FIFO of a ...... PMputdata(3P) PDR register DEVpipe put write ..... DEVpipe put(3S) PDR register DEVpixel put send ...... DEVpixel put(3S) pipe and pixel nodes ...... DEVrun(3H) pipe board ...... DEVfifo\_reset(3S) pipe board to operate in parallel ..... DEVfifo parallel(3S) pipe board to operate in serial ..... DEVfifo serial(3S) pipe DSP DEVpipe get read ..... DEVpipe get(3S) pipe DSP DEVpipe get msg ..... DEVpipe get msg(3S) pipe DSP DEVpipe get pir ..... DEVpipe get pir(3S) pipe DSP DEVpipe\_read ..... DEVpipe\_read(3S) pipe DSP ..... DEVpipe\_write(3S) pipe DSP's PDR register ..... DEVpipe put(3S) pipe feedback FIFO /read ..... DEVfifo read(3S) pipe FIFO DEVfifo write write ..... DEVfifo write(3S) pipe node /parameter count, and ..... PMcopycmd(3P) pipe node /get opcode and parameter ..... PMgetop(3P) pipe node /parameter count, and ..... PMputcmd(3P) pipe node PMputdata write ..... PMputdata(3P) pipe node /opcode and parameter ...... PMputop(3P) pipe node FIFO ..... PMgetdata(3P)

**DEVtools Reference Manual** 

output FIFO of a/ PMputop write from input to/ PMcopycmd copy parameters to the/ PMputcmd write device DEVinit /configure a pipe board to /configure a pipe board to buffer PMputpix **PMoutpir** output FIFO PMfb on direct printf formatted output commands to the regular /count, and data from input to /count, and parameters to the PMputdata write parameters to the opcode and parameter count to the DEVpixel overlay update register DEVpixel\_mode\_overlay set PMoverlay turn /send and receive data decrement references to a to a section of/ PMgetzaddr load a /PMset hireg reserve DRAM and and Z/ /PMxlate macros to manipulate a pipe board to operate in input to/ PMcopycmd copy opcode, the output/ PMputcmd write opcode. a pipe node PMgetop get opcode and of a pipe/ PMputop write opcode and /write opcode, parameter count, and pipe node PMputdata write a block of data to a pipe DSP's a block of data to a pixel DSP's PMfdiv DEVrun begin execution of all DEVfifo reset resets all FIFOs on a mode DEVfifo parallel configure a mode DEVfifo\_serial configure a a stream of bytes from the PIR of a read a message from the PIR of a

input FIFO of a pipe/ PMgetop get

read the PIR register of a reads a block of memory from a DEVpipe\_write write a buffer to a /write a block of data to a a block of four byte values from a a block of four byte values from a data from input to output FIFO of a count from input FIFO of a parameters to the output FIFO of a count to the output FIFO of a PMgetdata get data from a

pipe node processorDEVpipe_halt(3Spipe nodes /load a Pixel MachineDEVpipe_boot(3Spipe nodes /begin executionDEVpipe_run(3Spipe system /switch primaryDEVswap_pipe(3Hpipes and read commands back/DEVswap_pipe(3Hpipes of a dual pipe systemDEVswap_pipe(3HPIR of a pipe DSPDEVpipe_getPIR of a pipe DSPDEVpipe_get_msgPIR registerDEVpixel_get_msgPIR registerDEVpixel_get_msgPIR registerDEVpixel_get_msgPIR register of a pipe DSPDEVpixel_get_msg(3SPIR register of a pipe DSPDEVpixel_get_msg(3SPIR register of a pipe DSPDEVpixel_get_pir(3Spixel PMpixaddrPMpixaddrpixel board mode registerDEVpixel_get_pir(3Spixel DSPDEVpixel_get_pirpixel DSPDEVpixel_get_pir(3Spixel DSPDEVpixel_get_pir(3Spixel DSPDEVpixel_get_pir(3Spixel DSPDEVpixel_get_pir(3Spixel DSPDEVpixel_get_pir(3Spixel DSPDEVpixel_get_pir(3Spixel DSPDEVpixel_get_pir(3Spixel DSPDEVpixel_get_pir(3Spixel DSPDEVpixel_get_pir(3Spixel DSP's PDR registerDEVpixel_get_mit(3Spixel DSP's PIR registerDEVpixel_get_si(3Spixel DSP's PIR registerDEVpixel_get_ms(3S)pixel from buffer 0PMv03Xpixel from buffer 1PMv(3Xpixel from the current bufferPMgetj(3Xpixel from the current bufferPMgetj(3Xpixel from
pipe nodes/load a Pixel MachineDEVpipe_boot(3S)pipe nodes/begin executionDEVpipe_run(3S)pipe systemDEVswap_pipe(3H)pipelines and read commands back/DEVswap_pipe(3H)pipes of a dual pipe systemDEVswap_pipe(3H)PIR of a pipe DSPDEVpipe_getPIR of a pipe DSPDEVpipe_get_msgPIR registerDEVpixel_get read aDEVpixel_getDEVpixel_get_msgPIR registerDEVpixel_get_msgPIR registerDEVpixel_get_msgPIR register of a pipe DSPDEVpipe_get_pir(3S)PIR register of a pipe DSPDEVpixel_get_pir(3S)pixel PMpixaddrPMoutpir(3N)PIR register of a pipe DSPDEVpixel_get_pir(3S)pixel board mode registerDEVpixel_get_pir(3S)pixel DSPDEVpixel_get_pir(3S)pixel DSPDEVpixel_get_mic(3S)pixel DSPDEVpixel_get_mic(3S)pixel DSPDEVpixel_get_mic(3S)pixel DSP's PIR registerDEVpixel_get_mic(3S)pixel from buffer 0PMvQst(3X)pixel from buffer 1PMvQst(3X)pixel from the current
pipe nodes/begin executionDEVpipe_run(3Spipe system/switch primaryDEVswap_pipe(3Hpipelines and read commands back/DEVswap_pipe(3Hpipes of a dual pipe systemDEVswap_pipe(3HPIR of a pipe DSPDEVpipe_getDEVpipe_get(3SPIR of a pipe DSPDEVpipe_get_msgDEVpipe_get_msg(3SPIR registerDEVpixel_get read aDEVpixel_get(3SPIR registerDEVpixel_get_msgDEVpixel_get_msg(3SPIR registerDEVpixel_get_msgDEVpixel_get_msg(3SPIR register of a pipe DSPDEVpipe_get_pir(3Spixel PMpixaddrPMoutpir(3NPIR register of a pixel DSPDEVpixel_get_pir(3Spixel board mode registerDEVpixel_mode_init(3Xpixel boshDEVpixel_get_pirpixel DSPDEVpixel_get_pir(3Spixel DSPDEVpixel_get_siterpixel DSPDEVpixel_get_siterpixel DSPDEVpixel_get_siterpixel DSPDEVpixel_get_siterpixel DSP's PIR registerDEVpixel_get_siterpixel from buffer 0PMv0get(3Xpixel from buffer 1PMv(3X
pipe system /switch primaryDEVswap_pipe(3Hpipelines and read commands back/DEVwirte(3Hpipes of a dual pipe systemDEVswap_pipe(3HPIR of a pipe DSPDEVpipe_getPIR of a pipe DSPDEVpipe_get_msgPIR registerDEVpixel_get read aPIR registerDEVpixel_get_msgPIR registerDEVpixel_get_msgPIR registerDEVpixel_get_msgPIR registerDEVpixel_get_msgPIR register of a pipe DSPDEVpipe_get_pir(3S)PIR register of a pipe DSPDEVpixel_get_pir(3S)pixel PMpixaddrPMpixaddrpixel board mode registerDEVpixel_mode_init(3S)pixel DSPDEVpixel_get_pir(3S)pixel DSPDEVpixel_get_mir(3S)pixel DSPDEVpixel_get_mir(3S)pixel DSPDEVpixel_get_pir(3S)pixel DSPDEVpixel_get_mir(3S)pixel DSPPIR registerDSPDEVpixel_get_mir(3S)<
pipelines and read commands back/
pipes of a dual pipe systemDEVswap_pipe(3HPIR of a pipe DSPDEVpipe_getDEVpipe_get(3S)PIR of a pipe DSPDEVpipe_get_msgDEVpipe_get_msg(3S)PIR registerDEVpixel_get read aDEVpixel_get_get(3S)PIR registerDEVpixel_get_msgDEVpixel_get_msg(3S)PIR registerDEVpixel_get_msgDEVpixel_get_msg(3S)PIR registerDEVpixel_get_msgDEVpixel_get_msg(3S)PIR register of a pipe DSPDEVpipe_get_pir(3S)pixel PMpixaddrPMpixaddrPMpixaddr(3X)pixel board mode registerDEVpixel_mode_init(3S)pixel buffersPMswapbuff(3X)pixel DSPDEVpixel_get_pirpixel DSPDEVpixel_readpixel DSPDEVpixel_readpixel DSPDEVpixel_readpixel DSPDEVpixel_readpixel DSPDEVpixel_readpixel DSPDEVpixel_readpixel DSP's PDR registerDEVpixel_write(3S)pixel DSP's PIR registerDEVpixel_get_msg(3S)pixel from buffer 0PMv0get(3X)pixel from buffer 1PMv(3X)pixel from the current bufferPMgetpix(3X)pixel from the current bufferPMgetg(3X)pixel from the current bufferDEVpixel_get pixel(3S)pixel from the current bufferDEVpixelpixel from the current bufferDEVpixelpixel from the current bufferDEVpixelpixel from the current bufferDEVpixelpixel from the frame bufferDEVpixelpixel from the frame bufferDEVpixel
PIR of a pipe DSP       DEVpipe_get       DEVpipe_get(3S)         PIR of a pipe DSP       DEVpipe_get_msg       DEVpipe_get_msg(3S)         PIR register       DEVpixel_get read a       DEVpixel_get_msg(3S)         PIR register       DEVpixel_get_msg       DEVpixel_get_msg(3S)         PIR register       DEVpixel_get_msg       DEVpixel_get_msg(3S)         PIR register       DEVpixel_get_msg       DEVpixel_get_msg(3S)         PIR register of a pipe DSP       DEVpixel_get_pir(3S)         pixel       PMpixaddr       PMoutpir(3N)         pixel PMpixaddr       PMpixaddr(3X)         pixel board mode register       DEVpixel_mode_init(3S)         pixel buffers       PMswapbuff(3X)         pixel DSP       DEVpixel_get_pir         DSP       DEVpixel_read         pixel DSP       DEVpixel_read         pixel DSP       DEVpixel_read         pixel DSP       DEVpixel_read         pixel DSP       DEVpixel_read(3S)         pixel DSP       DEVpixel_get_msg(3S)         pixel DSP's PDR register       DEVpixel_pet(3S)         pixel DSP's PIR register       DEVpixel_get_msg(3S)         pixel from buffer 0       PMv0get(3X)         pixel from buffer 1       PMv0get(3X)         pixel from t
PIR of a pipe DSP       DEVpipe_get_msg       DEVpipe_get_msg(3S)         PIR register       DEVpixel_get_get_get_get_get_get_get_get_get_get
PIR register       DEVpixel_get read a       DEVpixel_get_ds(3S)         PIR register       DEVpixel_get_msg       DEVpixel_get_msg(3S)         PIR register       PMoutpir(3N)         PIR register of a pipe DSP       DEVpipe_get_pir(3S)         pixel       PMpixaddr       PMpixaddr(3X)         pixel       PMpixaddr       PMpixaddr(3X)         pixel board mode register       DEVpixel_mode_init(3S)         pixel buffers       PMswapbuff(3X)         pixel DSP       DEVpixel_get_pir         DEVpixel_get_pir       DEVpixel_get_pir(3S)         pixel DSP       DEVpixel_read         DSP       DEVpixel_read         pixel DSP       DEVpixel_read         pixel DSP's PDR register       DEVpixel_write(3S)         pixel DSP's PIR register       DEVpixel_get_msg(3S)         pixel DSP's PIR register       DEVpixel_get_msg(3S)         pixel DSP's PIR register       DEVpixel_get_msg(3S)         pixel from buffer 0       PMv00get(3X)         pixel from buffer 1       PMv(3X)         pixel from the current buffer       PMgetpix(3X)         pixel from the current buffer       PMgetg(3X)         pixel from the frame buffer       DEVpixel_get pixel(3S)
PIR register       DEVpixel_get_msg       DEVpixel_get_msg(3S)         PIR register       PMoutpir(3N)         PIR register of a pipe DSP       DEVpipe_get_pir(3S)         PIR register of a pixel DSP       DEVpixel_get_pir(3S)         pixel PMpixaddr       PMpixaddr(3X)         pixel board mode register       DEVpixel_mode_init(3S)         pixel buffers       PMswapbuff(3X)         pixel DSP       DEVpixel_get_pir         DEV DEVpixel_get_pir       DEVpixel_get_pir(3S)         pixel DSP       DEVpixel_read         DSP       DEVpixel_read         pixel DSP       DEVpixel_read         DSP       DEVpixel_read         pixel DSP's PDR register       DEVpixel_write(3S)         pixel DSP's PIR register       DEVpixel_get_msg(3S)         pixel DSP's PIR register       DEVpixel_get_msg(3S)         pixel DSP's PIR register       DEVpixel_get(3S)         pixel from buffer 0       PMv0get(3X)         pixel from buffer 1       PMv(3X)         pixel from the current buffer       PMgetpix(3X)         pixel from the current buffer       PMgetg(3X)         pixel from the frame buffer       DEVpixel_get pixel(3S)
PIR register       PMoutpir(3N)         PIR register of a pipe DSP       DEVpipe_get_pir(3S)         PIR register of a pixel DSP       DEVpixel_get_pir(3S)         pixel PMpixaddr       PMpixaddr(3X)         pixel board mode register       DEVpixel_mode_init(3S)         pixel DSP       DEVpixel_get_pir(3S)         pixel DSP       DEVpixel_get_pir(3S)         pixel DSP       DEVpixel_get_pir(3S)         pixel DSP       DEVpixel_get_pir(3S)         pixel DSP       DEVpixel_read         DEV       DEVpixel_read         pixel DSP       DEVpixel_read         pixel DSP       DEVpixel_read         pixel DSP       DEVpixel_read         pixel DSP's PDR register       DEVpixel_get_03S)         pixel DSP's PIR register       DEVpixel_get_msg(3S)         pixel DSP's PIR register       DEVpixel_get_msg(3S)         pixel from buffer 0       PMv0get(3X)         pixel from buffer 1       PMv(3X)         pixel from the current buffer       PMgetpix(3X)         pixel from the current buffer       PMgetpix(3X)         pixel from the current buffer       DEVget_pixel(3S)         pixel from the current buffer       DEVget_pixel(3S)
PIR register of a pipe DSP       DEVpipe_get_pir(3S)         PIR register of a pixel DSP       DEVpixel_get_pir(3S)         pixel PMpixaddr       PMpixaddr(3X)         pixel board mode register       DEVpixel_mode_init(3S)         pixel DSP       DEVpixel_get_pir         pixel DSP       DEVpixel_get_pir(3S)         pixel DSP       DEVpixel_get_pir(3S)         pixel DSP       DEVpixel_get_pir(3S)         pixel DSP       DEVpixel_get_pir(3S)         pixel DSP       DEVpixel_read         pixel DSP       DEVpixel_read         pixel DSP's PDR register       DEVpixel_put(3S)         pixel DSP's PIR register       DEVpixel_get_msg(3S)         pixel DSP's PIR register       DEVpixel_get_msg(3S)         pixel from buffer 0       PMv0get(3X)         pixel from buffer 1       PMv(3X)         pixel from the current buffer       PMgetpix(3X)         pixel from the current buffer       PMgetpix(3X)         pixel from the current buffer       DEVget pixel(3S)
PIR register of a pipe DSP       DEVpipe_get_pir(3S)         PIR register of a pixel DSP       DEVpixel_get_pir(3S)         pixel PMpixaddr       PMpixaddr(3X)         pixel board mode register       DEVpixel_mode_init(3S)         pixel DSP       DEVpixel_get_pir         pixel DSP       DEVpixel_get_pir(3S)         pixel DSP       DEVpixel_get_pir(3S)         pixel DSP       DEVpixel_get_pir(3S)         pixel DSP       DEVpixel_get_pir(3S)         pixel DSP       DEVpixel_read         pixel DSP       DEVpixel_read         pixel DSP's PDR register       DEVpixel_put(3S)         pixel DSP's PIR register       DEVpixel_get_msg(3S)         pixel DSP's PIR register       DEVpixel_get_msg(3S)         pixel from buffer 0       PMv0get(3X)         pixel from buffer 1       PMv(3X)         pixel from the current buffer       PMgetpix(3X)         pixel from the current buffer       PMgetpix(3X)         pixel from the current buffer       DEVget pixel(3S)
PIR register of a pixel DSP       DEVpixel_get_pir(3S)         pixel PMpixaddr       PMpixaddr(3X)         pixel board mode register       DEVpixel_mode_init(3S)         pixel buffers       PMswapbuff(3X)         pixel DSP       DEVpixel_get_pir         DEV DEVpixel_get_pir       DEVpixel_get_pir(3S)         pixel DSP       DEVpixel_read         DEV       DEVpixel_read(3S)         pixel DSP       DEVpixel_read(3S)         pixel DSP's PDR register       DEVpixel_put(3S)         pixel DSP's PIR register       DEVpixel_get_msg(3S)         pixel DSP's PIR register       DEVpixel_get_msg(3S)         pixel from buffer 0       PMv0get(3X)         pixel from buffer 1       PMv(3X)         pixel from the current buffer       PMgetpix(3X)         pixel from the current buffer       PMgetpix(3X)         pixel from the current buffer       DEVget pixel(3S)         pixel from the current buffer       DEVget pixel(3S)
pixel       PMpixaddr       PMpixaddr(3X         pixel       board       mode       register       DEVpixel_mode_init(3S         pixel       DSP       DEVpixel_get_pir       DEVpixel_get_pir(3S)         pixel       DSP       DEVpixel_read       DEVpixel_read(3S)         pixel       DSP       DEVpixel_read       DEVpixel_write(3S)         pixel       DSP       DEVpixel_read       DEVpixel_urite(3S)         pixel       DSP's       PDR register       DEVpixel_put(3S)         pixel       DSP's       PIR register       DEVpixel_get_msg(3S)         pixel       DSP's       PIR register       DEVpixel_get_msg(3S)         pixel       from buffer 0       PMv0get(3X)         pixel from buffer 1       PMv(3X)       Pixel from buffer 1       PMv(3X)         pixel from the current buffer       PMgetpix(3X)       Pixel from the current buffer       PMgetpix(3X)         pixel from the current buffer       DEVget pixel(3X)       Pixel from the frame buffer       DEVget pixel(3X)
pixel board mode register       DEVpixel_mode_init(3S         pixel buffers       PMswapbuff(3X         pixel DSP       DEVpixel_get_pir         DEVDixel_get_pir       DEVpixel_get_pir(3S)         pixel DSP       DEVpixel_read         pixel DSP       DEVpixel_read         pixel DSP       DEVpixel_write(3S)         pixel DSP's PDR register       DEVpixel_put(3S)         pixel DSP's PIR register       DEVpixel_get_msg(3S)         pixel DSP's PIR register       DEVpixel_get_msg(3S)         pixel from buffer 0       PMv0get(3X)         pixel from buffer 1       PMv(3X)         pixel from the current buffer       PMgetpix(3X)         pixel from the current buffer       PMgetpix(3X)         pixel from the current buffer       DEVget pixel(3S)         pixel from the current buffer       DEVget pixel(3S)
pixel buffers       PMswapbuff(3X         pixel DSP       DEVpixel_get_pir       DEVpixel_get_pir(3S)         pixel DSP       DEVpixel_read       DEVpixel_read(3S)         pixel DSP       DEVpixel_read       DEVpixel_write(3S)         pixel DSP's PDR register       DEVpixel_put(3S)         pixel DSP's PIR register       DEVpixel_get_msg(3S)         pixel DSP's PIR register       DEVpixel_get_msg(3S)         pixel from buffer 0       PMv0get(3X)         pixel from buffer 1       PMv(3X)         pixel from the current buffer       PMgetpix(3X)         pixel from the current buffer       PMgetpix(3X)         pixel from the current buffer       DEVget pixel(3S)
pixel DSP       DEVpixel get_pir       DEVpixel get_pir(3S)         pixel DSP       DEVpixel read       DEVpixel read(3S)         pixel DSP       DEVpixel write(3S)       DEVpixel write(3S)         pixel DSP's PDR register       DEVpixel put(3S)         pixel DSP's PIR register       DEVpixel get_msg(3S)         pixel DSP's PIR register       DEVpixel get_msg(3S)         pixel from buffer 0       PMv0get(3X)         pixel from buffer 1       PMv(3X)         pixel from the current buffer       PMgetpix(3X)         pixel from the current buffer       PMgetpix(3X)         pixel from the current buffer       DEVget pixel(3S)
pixel DSP       DEVpixel read       DEVpixel read(3S)         pixel DSP       DEVpixel write(3S)         pixel DSP's PDR register       DEVpixel put(3S)         pixel DSP's PIR register       DEVpixel get(3S)         pixel DSP's PIR register       DEVpixel get(3S)         pixel DSP's PIR register       DEVpixel get(3S)         pixel from buffer 0       PMv0get(3X)         pixel from buffer 1       PMv(3X)         pixel from the current buffer       PMgetpix(3X)         pixel from the current buffer       PMget(3X)         pixel from the current buffer       DEVget pixel(3S)
pixel DSP       DEVpixel_write(3S)         pixel DSP's PDR register       DEVpixel_put(3S)         pixel DSP's PIR register       DEVpixel_get(3S)         pixel DSP's PIR register       DEVpixel_get(3S)         pixel DSP's PIR register       DEVpixel_get(3S)         pixel from buffer 0       PMv0get(3X)         pixel from buffer 1       PMv(3X)         pixel from the current buffer       PMgetpix(3X)         pixel from the current buffer       PMgetpix(3X)         pixel from the current buffer       DEVget pixel(3X)         pixel from the current buffer       DEVget pixel(3X)
pixel DSP's PDR register
pixel DSP's PIR register
pixel DSP's PIR register
pixel DSP's PIR register
pixel from buffer 0 PMv0get(3X pixel from buffer 1 PMv0get(3X pixel from the current buffer PMgetpix(3X pixel from the current buffer PMgetgix(3X pixel from the frame buffer DEVget pixel(3S
pixel from buffer 1
pixel from the current bufferPMgetpix(3X pixel from the current bufferPMqget(3X pixel from the frame buffer
pixel from the current buffer PMqget(3X pixel from the frame buffer
pixel from the frame buffer DEVget pixel(3S
pixel from the frame buffer DEVget_pixel(3S
Pixel Machine devdisp devdisp(1
Pixel Machine /download an DEVput_scan_line(3H
Pixel Machine /download an
Pixel machine available to a user/
Pixel machine available to a user/
Pixel machine available to a user/
Pixel machine available to a user/       DEVopen(3S         Pixel Machine code that uses the/       devprint(1         Pixel Machine device       DEVexit(3H         Pixel Machine device       DEVexit(3H         Pixel Machine device       DEVinit(3H)
Pixel machine available to a user/       DEVopen(3S)         Pixel Machine code that uses the/       devprint(1         Pixel Machine device       DEVexit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine executable into       DEVpipe_boot(3S)
Pixel machine available to a user/       DEVopen(3S)         Pixel Machine code that uses the/       devprint(1         Pixel Machine device       DEVexit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpixel boot(3S)
Pixel machine available to a user/       DEVopen(3S)         Pixel Machine code that uses the/       devprint(1         Pixel Machine device       DEVexit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpixel_boot(3S)         Pixel Machine image header from a       DEVget image header(3S)
Pixel machine available to a user/       DEVopen(3S)         Pixel Machine code that uses the/       devprint(1         Pixel Machine device       DEVexit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpixel_boot(3S)         Pixel Machine image header from a       DEVget_image_header(3S)         Pixel Machine image header to a       DEVput image_header(3S)
Pixel machine available to a user/       DEVopen(3S)         Pixel Machine code that uses the/       devprint(1         Pixel Machine device       DEVexit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpixel_boot(3S)         Pixel Machine image header from a       DEVget_image_header(3S)         Pixel Machine image header to a       DEVput_image_header(3S)         Pixel Machine locks       DEVlock(3S)
Pixel machine available to a user/       DEVopen(3S)         Pixel Machine code that uses the/       devprint(1         Pixel Machine device       DEVexit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpixel_boot(3S)         Pixel Machine image header from a       DEVget_image_header(3S)         Pixel Machine image header to a       DEVput_image_header(3S)         Pixel Machine locks       DEVlock(3S)         Pixel Machine locks       DEVlock(3S)         Pixel Machine program /to the host       PMhost exit(3N)
Pixel machine available to a user/       DEVopen(3S)         Pixel Machine code that uses the/       devprint(1         Pixel Machine device       DEVexit(3H)         Pixel Machine device       DEVopen(3S)         Pixel Machine device       DEVopen(3S)         Pixel Machine device       DEVoint(3H)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpixel_boot(3S)         Pixel Machine image header from a       DEVget_image_header(3S)         Pixel Machine image header to a       DEVput_image_header(3S)         Pixel Machine locks       DEVlock(3S)         Pixel Machine program /to the host       PMhost_exit(3N)         Pixel Machine programs using       devcc(1)
Pixel machine available to a user/       DEVopen(3S)         Pixel Machine code that uses the/       devprint(1         Pixel Machine device       DEVexit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpixel_boot(3S)         Pixel Machine image header from a       DEVget_image_header(3S)         Pixel Machine image header to a       DEVput_image_header(3S)         Pixel Machine locks       DEVlock(3S)         Pixel Machine program /to the host       PMhost_exit(3N)         Pixel Machine programs using       devcc(1)
Pixel machine available to a user/       DEVopen(3S)         Pixel Machine code that uses the/       devprint(1         Pixel Machine device       DEVexit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpixel_boot(3S)         Pixel Machine image header from a       DEVget_image_header(3S)         Pixel Machine image header to a       DEVput_image_header(3S)         Pixel Machine locks       DEVlock(3S)         Pixel Machine program /to the host       PMhost_exit(3N)         Pixel Machine programs using       devcc(1)
Pixel machine available to a user/       DEVopen(3S)         Pixel Machine code that uses the/       devprint(1)         Pixel Machine device       DEVexit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpixel_boot(3S)         Pixel Machine image header from a       DEVget_image_header(3S)         Pixel Machine image header to a       DEVput_image_header(3S)         Pixel Machine locks       DEVlock(3S)         Pixel Machine program /to the host       PMhost_exit(3N)         Pixel Machine programs using       devcc(1)         Pixel Machine to a file       devsave(1)         Pixel Machine to a file       DEVwrite(3H)
Pixel machine available to a user/       DEVopen(3S)         Pixel Machine code that uses the/       devprint(1)         Pixel Machine device       DEVexit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpixel_boot(3S)         Pixel Machine image header from a       DEVget_image_header(3S)         Pixel Machine image header to a       DEVput_image_header(3S)         Pixel Machine locks       DEVlock(3S)         Pixel Machine program /to the host       PMhost_exit(3N)         Pixel Machine programs using       devcc(1)         Pixel Machine to a file       devsave(1)         Pixel Machines pipelines and read       DEVwrite(3H)         Pixel Machines pipelines and read       DEVwrite(3H)
Pixel machine available to a user/       DEVopen(3S)         Pixel Machine code that uses the/       devprint(1)         Pixel Machine device       DEVexit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVgit
Pixel machine available to a user/       DEVopen(3S)         Pixel Machine code that uses the/       devprint(1)         Pixel Machine device       DEVexit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpixel_boot(3S)         Pixel Machine image header from a       DEVget_image_header(3S)         Pixel Machine image header to a       DEVput_image_header(3S)         Pixel Machine locks       DEVlock(3S)         Pixel Machine program /to the host       DEVlock(3S)         Pixel Machine programs using       devcc(1)         Pixel Machine to a file       devsave(1)         Pixel Machine to a file       DEVwrite(3H)         Pixel memory without subscreens /or       PMgetrow(3X)         pixel mode register       DEVpixel_mode_overlay(3S)         pixel node DEVread z read a buffer       DEVpixel_mode_fored_2(3S)
Pixel machine available to a user/       DEVopen(3S)         Pixel Machine code that uses the/       devprint(1)         Pixel Machine device       DEVexit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpixel_boot(3S)         Pixel Machine image header from a       DEVget_image_header(3S)         Pixel Machine image header to a       DEVput_image_header(3S)         Pixel Machine locks       DEVlock(3S)         Pixel Machine program /to the host       PMhost_exit(3H)         Pixel Machine programs using       devcc(1)         Pixel Machine programs using       devcc(1)         Pixel Machine to a file       devsave(1)         Pixel Machine to a file       DEVwite(3H)         Pixel Machine to a file       devca(2)         Pixel Machine to a file       DEVwite(3H)         pixel mode register       DEVpixel_mo
Pixel machine available to a user/       DEVopen(3S)         Pixel Machine code that uses the/       devprint(1)         Pixel Machine device       DEVexit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpixel_boot(3S)         Pixel Machine image header from a       DEVget_image_header(3S)         Pixel Machine image header to a       DEVput_image_header(3S)         Pixel Machine program /to the host       DEVlock(3S)         Pixel Machine programs using       devcc(1)         Pixel Machine programs using       devcc(1)         Pixel Machine to a file       devsave(1)         Pixel Machine spipelines and read       DEVpixel_mode_overlay(3S)         pixel memory without subscreens /or       PMgetrow(3X)         pixel mode register       DEVpixel_mode_overlay(3S)         pixel node DEVread_z read a buffer       DEVpixel_doverlay(3S)         pixel node DEVread_z read a buffer       DEVpixel id write(3S)         pixel node DEVread_z read a buffer       DEVpixel id write(3S)         pixel node DSP's memory /a node id       DEVpixel id write(3S)
Pixel machine available to a user/       DEVopen(3S)         Pixel Machine code that uses the/       devprint(1)         Pixel Machine device       DEVexit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpixel_boot(3S)         Pixel Machine image header from a       DEVget_image_header(3S)         Pixel Machine image header to a       DEVput_image_header(3S)         Pixel Machine program /to the host       DEVlock(3S)         Pixel Machine program /to the host       DEVlock(3S)         Pixel Machine programs using       devcc(1)         Pixel Machine programs using       devcc(1)         Pixel Machine programs using       DEVwrite(3H)         Pixel Machines pipelines and read       DEVwrite(3H)         pixel mode register       DEVpixel_mode_overlay(3S)         pixel node DEVread_z read a buffer       DEVread_z(3S)         pixel node DEVread_z read a buffer       DEVwrite
Pixel machine available to a user/       DEVopen(3S)         Pixel Machine code that uses the/       devprint(1)         Pixel Machine device       DEVexit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpixel_boot(3S)         Pixel Machine image header from a       DEVget_image_header(3S)         Pixel Machine image header to a       DEVput_image_header(3S)         Pixel Machine program /to the host       DEVlock(3S)         Pixel Machine program /to the host       PMhost_exit(3H)         Pixel Machine programs using       devcc(1)         Pixel Machine programs using       devcc(1)         Pixel Machine programs using       DEVwrit(3H)         Pixel Machine programs using       devcc(1)         Pixel Machine programs using       DEVwrit(3H)         Pixel Machine spipelines and read       DEVwrite(3H)         pixel mode register       DEVpixel_mode_overlay(3S)         pixel node DEVread_z read a buffer       DEVpixel_dow/da)         pixel node DEVread_z read a buffer       DEVwrite_3(A)         pixel node DSP's memory /a node id
Pixel machine available to a user/       DEVopen(3S)         Pixel Machine code that uses the/       devprint(1)         Pixel Machine device       DEVexit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpixel_boot(3S)         Pixel Machine image header from a       DEVget_image_header(3S)         Pixel Machine image header to a       DEVput_image_header(3S)         Pixel Machine program /to the host       DEVlock(3S)         Pixel Machine program /to the host       DEVlock(3S)         Pixel Machine programs using       devcc(1)         Pixel Machine programs using       devcc(1)         Pixel Machine programs using       DEVwrite(3H)         Pixel Machines pipelines and read       DEVwrite(3H)         pixel mode register       DEVpixel_mode_overlay(3S)         pixel node DEVread_z read a buffer       DEVread_z(3S)         pixel node DEVread_z read a buffer       DEVwrite
Pixel Machine /dewpload an DEVout scap line/3
pixel from the frame buffer
pixel from the frame buffer DEVget_pixel(3S
pixel from the frame buffer DEVget pixel(3S
pixel from the frame buffer DEVget pixel(3S
pixel from the current buffer PMqget(3X pixel from the frame buffer
pixel from the current buffer PMqget(3X pixel from the frame buffer
pixel from the current buffer PMqget(3X pixel from the frame buffer
pixel from the current buffer PMqget(3X pixel from the frame buffer
pixel from the current buffer PMqget(3X pixel from the frame buffer
pixel from the current buffer PMqget(3X pixel from the frame buffer
pixel from the current buffer PMqget(3X pixel from the frame buffer
pixel from the current buffer PMqget(3X pixel from the frame buffer
pixel from the current buffer PMqget(3X pixel from the frame buffer
pixel from the current buffer PMqget(3X pixel from the frame buffer
pixel from the frame buffer DEVget pixel(3S
pixel from the frame buffer DEVget pixel(3S
pixel from the frame buffer DEVget pixel(3S
pixel from the frame buffer DEVget pixel(3S
pixel from the frame buffer DEVget pixel(3S
pixel from the frame buffer DEVget pixel(3S
pixel from the frame buffer DEVget pixel(3S
pixel from the frame buffer DEVget pixel(3S
pixel from the frame buffer DEVget pixel(3S
pixel from the frame buffer DEVget pixel(3S
pixel from the frame buffer DEVget pixel(3S
pixel from the current buffer PMqget(3X pixel from the frame buffer
pixel from the current buffer PMqget(3X pixel from the frame buffer
pixel from the current buffer PMqget(3X pixel from the frame buffer
pixel from the current buffer PMqget(3X pixel from the frame buffer
pixel from the current buffer PMqget(3X pixel from the frame buffer
pixel from the current buffer PMqget(3X pixel from the frame buffer
pixel from the current buffer PMqget(3X pixel from the frame buffer
pixel from the current buffer PMqget(3X pixel from the frame buffer
pixel from the frame buffer DEVget pixel(3S
pixel from the frame buffer DEVget pixel(3S
pixel from the frame buffer DEVget pixel(3S
pixel from the frame buffer DEVget pixel(3S
pixel from the frame buffer DEVget pixel(3S
pixel from the frame buffer DEVget pixel(3S
pixel from the frame buffer
Pixel Machine DEVclose(3S
Pixel Machine devdiso devdiso(1
Pixel Machine devdisp devdisp(1
I
F(
Pixel Machine /download an DEVout scan line/3H
Pixel Machine /download an DEVput_scan_line(3H
Pixel Machine /download an DEVput_scan_line(3H
Pixel Machine /download an DEVput_scan_line(3H
Pixel Machine /download an
Pixel Machine /download an
Pixel machine available to a user/ DEVopen(3S
Pixel machine available to a user/ DEVopen(3S
Pixel machine available to a user/ DEVopen(3S
Pixel machine available to a user/
Pixel machine available to a user/       DEVopen(3S)         Pixel Machine code that uses the/       devprint(1         Pixel Machine device       DEVexit(3H)         Pixel Machine device       DEVexit(3H)         Pixel Machine device       DEVinit(3H)
Pixel machine available to a user/       DEVopen(3S)         Pixel Machine code that uses the/       devprint(1         Pixel Machine device       DEVexit(3H)         Pixel Machine device       DEVexit(3H)         Pixel Machine device       DEVinit(3H)
Pixel machine available to a user/       DEVopen(3S)         Pixel Machine code that uses the/       devprint(1         Pixel Machine device       DEVexit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine executable into       DEVpipe_boot(3S)
Pixel machine available to a user/       DEVopen(3S)         Pixel Machine code that uses the/       devprint(1         Pixel Machine device       DEVexit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpipe_boot(3S)
Pixel machine available to a user/       DEVopen(3S)         Pixel Machine code that uses the/       devprint(1         Pixel Machine device       DEVexit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpipe_boot(3S)
Pixel machine available to a user/       DEVopen(3S)         Pixel Machine code that uses the/       devprint(1         Pixel Machine device       DEVexit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpixel_boot(3S)         Pixel Machine image header from a       DEVget image header(3S)
Pixel machine available to a user/       DEVopen(3S)         Pixel Machine code that uses the/       devprint(1         Pixel Machine device       DEVexit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpixel_boot(3S)         Pixel Machine image header from a       DEVget_image_header(3S)         Pixel Machine image header to a       DEVput_image_header(3S)
Pixel machine available to a user/       DEVopen(3S)         Pixel Machine code that uses the/       devprint(1         Pixel Machine device       DEVexit(3H)         Pixel Machine device       DEVinit(3H)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpipe_boot(3S)         Pixel Machine executable into       DEVpixel_boot(3S)         Pixel Machine image header from a       DEVget_image_header(3S)         Pixel Machine image header to a       DEVput image_header(3S)

DEVpipe halt halt a executable into specified set of of programs loaded into specified and alternate pipes of a dual /to write to the Pixel Machines /switch primary and alternate read a stream of bytes from the read a message from the stream of bytes from a pixel DSP's read a message from a pixel DSP's PMoutpir output a value to the DEVpipe get\_pir read the DEVpixel get pir read the generate a pointer to a specific DEVpixel mode init initialize PMswapbuff swap front and back read the PIR register of a read a block of memory from a DEVpixel write write a buffer to a /send a block of data to a /read a stream of bytes from a /read a message from a PMv0get read a PMv1get read a PMgetpix read a PMqget quick read of a DEVget\_pixel, DEVget\_pixels read a DEVclose closes the download an image from a file to a image or a portion of an image to a DEVopen, DEVopen system make a devprint a host server program for DEVexit halts processors, closes DEVinit opens and initializes specified set/ DEVpipe\_boot load a specified set/ DEVpixel\_boot load a file DEVget image header read the file DEVput image header write a DEVlock manage that signals the completion of a DEVtools devcc C compiler for devsave upload an image from a commands/ /macros to write to the write a scanline or scancolumn from /set overlay mode in the of bytes from the Z memory of a of bytes into the Z memory of a block to a reserved location in a PMgetcmd load command from a DEVpixel halt halt a executable into specified set of of programs loaded into specified

pixel nodes DEVrun	
pixel nodes to signal completion,	
pixel processor's flag registers	. DEvpixel_overlay(3S)
pixel processors to synchronize	PMpsync(3X)
pixel to buffer 0	
pixel to buffer 1	
pixel to the current buffer	
pixel to the current buffer	PMqput(3X)
pixels into the frame buffer	
PMapply apply a function to all	PMapply(3X)
PMavail_reg, PMset_lowreg,	
PMblock_reg, PMavail_reg,	
PMclear fill a rectangular region	
PMcolor_float macro that converts	
PMcolor_int macro that converts	
PMcommand data structure used for	
PMcommand data structure used for	
PMcopycmd copy opcode, parameter	
PMcopy_f fast but dangerous 32 bit	PMcopy_f(3X)
PMcopyftob copy front to back	PMcopyftob(3X)
PMcopy_s safe 32-bit DRAM or VRAM	PMcopy_s(3X)
PMcopy_v 32-bit copy with variable	PMcopy_v(3X)
PMcopyvtov copy blocks of VRAM	PMcopyvtov(3X)
PMcopyvtoz copy video RAM to DRAM	PMcopyvtoz(3X)
PMcopyztoz copy from one section of	PMcopyztoz(3X)
PMcos trigonometric function to	PMcos(3M)
PMdblbuff enable double buffering	PMdblbuff(3X)
PMdelay do nothing for a specified	PMdelay(3N)
PMdesc, PMxlate macros to	PMpagereg(3X)
PMenable enable processing of	PMenable(3N)
PMfb on direct output commands to	PMfb on(3P)
PMfdiv perform floating point	PMfdiv(3M)
PMflagled turn the PM_FLAG LED	
PMfloat color macro that converts	PMfloat color(3N)
PMfreezaddr decrement references to	PMfreezaddr(3X)
PMfxtoi map a linear function of x	
PMfxytoij map a linear function of	PMfxvtoii(3X)
PMfytoj map a linear function of y	PMfytoi(3X)
PMgetcmd load command from a pixel	PMgetcmd(3X)
PMgetcol, PMputrow, PMputcol read	PMaetrow(3X)
PMgetdata get data from a pipe node	
PMgetop get opcode and parameter	
PMgetpix read a pixel from the	
PMgetrow, PMgetcol, PMputrow,	
PMgetscan read a scanline from a	
PMgetzaddr load a page register and	
PMgetzbuf read a float value from	
PMgetzdesc, PMzdesc valid allocate	
PMhost exit send a message to the	
PMieee dsp convert IEEE float to	
PMihi map from screen space (xmax)	
PMilo map from screen space (xmax)	DMilo/2V
i mile map nom sereen space (xmm)	

**DEVtools Reference Manual** 

/update overlay mode in all PMpsync wait for all PMv0put write a PMv1put write a PMputpix output a PMqput quick write of a DEVput\_pixel, DEVput\_pixels write subscreens PMset hireg/ PMzbrk, PMblock reg, PMset\_lowreg, PMset\_hireg/ PMzbrk, of the screen internal color value to floating/ internal color value to an integer FIFO commands FIFO commands count, and data from input to/ D/VRAM copy

begin execution of all pipe and then call/ DEVwait exit wait for

copy increments

DRAM to another compute the cosine of an angle mode time manipulate page/ PMpagereg, selected system commands the regular output FIFO division on or off floating point value to internal/ a page register from screen space to processor/ x and y from screen space to/ from screen space to processor/ node FIFO or write a scanline or/ PMgetrow, **FIFO** count from input FIFO of a pipe/ current buffer PMputcol read or write a scanline/ subscreen return an address to a section of/ the Z buffer a DRAM block host that signals the completion/ DSP float to processor space (ihi) to processor space (ilo)

14

(

integer to an internal color value	PMint_color macro that converts an	
deinterleave a block	PMinterleave interleave or	
to processor space (jhi)	PMjhi map from screen space (ymax)	
to processor space (jlo)	PMjlo map from screen space (ymin)	
light sources	PMIdot specialized dot product for	PMIdot(3M)
longs to float	PMlong_dsp convert an array of	
data packet over serial links	PMmsg_exchange send and receive	PMmsg_exchange(3X)
pointer	PMmsg_setup set serial DMA input	
coordinate is in processor space	PMmyx test if a given screen space	PMmyx(3X)
coordinate is in processor space	PMmyy test if a given screen space	PMmyy(3X)
return its length	PMnorm normalize a 3D vector and	PMnorm(3M)
register	PMoutpir output a value to the PIR	PMoutpir(3N)
	PMoverlay turn overlay on or off	
to manipulate page registers used/	PMpagereg, PMdesc, PMxlate macros	
specific pixel	PMpixaddr generate a pointer to a	PMpixaddr(3X)
	PMpow power function	PMpow(3M)
processors to synchronize	PMpsync wait for all pixel	PMpsync(3X)
count, and parameters to the/	PMputcmd write opcode, parameter	PMputcmd(3P)
or/ PMgetrow, PMgetcol, PMputrow,	PMputcol read or write a scanline	PMgetrow(3X)
output FIFO of a pipe node	PMputdata write parameters to the	PMputdata(3P)
count to the output FIFO of a pipe/	PMputop write opcode and parameter	PMputop(3P)
current buffer	PMputpix output a pixel to the	
scanline or/ PMgetrow, PMgetcol,	PMputrow, PMputcol read or write a	
subscreen	PMputscan write a scanline to a	
the Z buffer	PMputzbuf write a float value to	
of DRAM to another	PMqcopyztoz copy from one section	
the current buffer	PMqget quick read of a pixel from	
the current buffer	PMqput quick write of a pixel to	
or off	PMrdyled turn the PM RDY LED on	
	PMrdyoff turn the ready signal off	PMrdyoff(3X)
/PMavail reg, PMset lowreg,	PMset hireg reserve DRAM and page/	
PMzbrk, PMblock reg, PMavail reg,	PMset lowreg, PMset hireg reserve/	
	PMsetsem set the semaphore	• •
	PMsin trigonometric function	
direction	PMsiodir set serial I/O link	
	PMsioinit initialize serial I/O	. ,
mode	PMsnglbuff disable double buffering	
	PMsqrt sqare root function	
buffer	PMswapback swap meaning of back	PMswapback(3)
pixel buffers	PMswapbuff swap front and back	,
the broadcast bus is granted	PMswap_pipe wait until control of	
the host	PMusermsg send a user message to	
	PMv0get read a pixel from buffer 0	
	PMv0put write a pixel to buffer 0	
	PMv1get read a pixel from buffer 1	• • • •
	PMv1put write a pixel to buffer 1	
vertical retrace	PMvsync synchronize and wait for	PMvsvnc(3X)
clear	PMwaitsem wait for semaphore to	
screen space	PMxat map subscreen coordinates to	
	PMx exp n integer power function	
registers used/ PMpagereg, PMdesc,	PMxlate macros to manipulate page	
screen space	PMyat map subscreen coordinates to	
	· · · · · · · · · · · · · · · · · · ·	·····,····,····,····,····

Permuted Index

a row	PMzaddr generate a ZRAM pointer to	PMzaddr(3X)
to a column	PMzaddrcol generate a ZRAM pointer	PMzaddrcol(3X)
PMset_lowreg, PMset_hireg reserve/	PMzbrk, PMblock_reg, PMavail_reg,	PMzbrk(3X)
PMgetzdesc,	PMzdesc valid allocate a DRAM block	PMgetzdesc(3X)
buffer	PMzget read a float from the z	
Z-buffer	PMzput write a float to the	
PMfdiv perform floating	point division	
format to the DSP32 floating	point format /floating-point	
internal color value to floating	point number /macro that converts	
/macro that converts floating	point value to internal color value	
PMmsg setup set serial DMA input	pointer	
PMzaddrcol generate a ZRAM	pointer to a column	
PMzaddr generate a ZRAM	pointer to a row	
PMpixaddr generate a	pointer to a specific pixel	
DEVpoll nodes	poll DSP processors for messages	
Machine /download an image or a	portion of an image to a Pixel	
PMpow	power function	
PMx exp n integer	power function	
dual pipe/ DEVswap pipe switch	primary and alternate pipes of a	
Pixel Machine code that uses the	print routines /server program for	
	print the node ID of a processor	
DEVpipe_id_print read and		
DEVpixel_id_print read and on host	print the node ID of a processor	
	printf formatted output conversion	
commands PMenable enable	processing of selected system	
DEVpipe_halt halt a pipe node	processor	
read and print the node ID of a	processor DEVpipe_id_print	
DEVpixel_halt halt a pixel node	processor	
read and print the node ID of a	processor DEVpixel_id_print	
given screen space coordinate is in	processor space PMmyx test if a	
given screen space coordinate is in	processor space PMmyy test if a	
function of x from screen space to	processor space i /map a linear	
of x and y from screen space to	processor space i and j /function	
map from screen space (xmax) to	processor space (ihi) PMihi	
map from screen space (xmin) to	processor space (ilo) PMilo	
function of y from screen space to	processor space j /map a linear	
map from screen space (ymax) to	processor space (jhi) PMjhi	
map from screen space (ymin) to	processor space (jlo) PMjlo	PMjlo(3X)
in the memory of one of the DSP	processors /the software semaphore	
	[	DEVrelease_pipe_semaphore(3H)
device DEVexit halts	processors, closes Pixel Machine	
/update overlay mode in all pixel	processor's flag registers	
DEVpoll_nodes poll DSP	processors for messages	
PMpsync wait for all pixel	processors to synchronize	
PMIdot specialized dot	product for light sources	
a Pixel machine available to a user	program /DEVopen_system make	DEVopen(3S)
the completion of a Pixel Machine	program /to the host that signals	PMhost_exit(3N)
uses the/ devprint a host server	program for Pixel Machine code that	
DEVpipe_run begin execution of	programs loaded into specified pipe/	DEVpipe_run(3S)
DEVpixel_run begin execution of	programs loaded into specified/	
devcc C compiler for Pixel Machine	programs using DEVtools	
current buffer PMqget	quick read of a pixel from the	PMqget(3X)
current buffer PMqput	quick write of a pixel to the	PMqput(3X)

**DEVtools Reference Manual** 

;

. . . . .

ideo	RAM	(3X)
ideo	RAM to DRAM	PMcopyvtoz(3X)
read	read a block of four byte values	DEVfifo read(3S)
read	read a block of memory from a pixel	
ad z	read a buffer of bytes from the Z	DEVread z(3S)
zget	read a float from the z buffer	PMzget(3X)
zbuf	read a float value from the Z	PMgetzbuf(3X)
msg	read a message from a pixel DSP's	DEVpixel get msg(3S)
msg	read a message from the PIR of a	DEVpipe get msg(3S)
Oget	read a pixel from buffer 0	
1get	read a pixel from buffer 1	PMv(3X)
etpix	read a pixel from the current	PMaetoix(3X)
ixels	read a pixel from the frame buffer	
scan	read a scanline from a subscreen	
	read a stream of bytes from a pixel	
_get	read a stream of bytes from the PIR	DEVpixel_get(00)
_get print	read and print the node ID of a	
print	read and print the node ID of a	
and	read commands back from the/ /write	DEV/write(30)
uick	read of a pixel from the current	DEVWINE(3H)
line	read one or more scan lines from a	
	read or write a scanline or/	DEvget_scan_ine(3H)
utcol	read or write a scaniine or/	DEVice selection (32)
map	read the color tables from video	DEVision and sin(35)
t_pir	read the PIR register of a pipe DSP	DEvpipe_get_pir(3S)
t_pir	read the PIR register of a pixel	DEVpixel_get_pir(3S)
ader	read the Pixel Machine image header	
		DEVget_image_header(3S)
read	reads a block of memory from a pipe	DEVpipe_read(3S)
bles	reads file of gamma calibration	DEVload_color_tables(3S)
n the	ready signal off	PMrdyoff(3X)
and	receive data packet over serial	PMmsg_exchange(3X)
fill a	rectangular region of the screen	PMclear(3X)
ment	references to a page register	PMtreezaddr(3X)
gular	region of the screen	PMclear(3X)
PDR	register DEVpipe_put write	
PIR	register /read a stream	
PIR	register DEVpixel_get_msg read	
node	register DEVpixel_mode_init	DEVpixel_mode_init(3S)
node	register DEVpixel_mode_overlay	DEVpixel_mode_overlay(3S)
PDR	register DEVpixel_put send a	DEVpixel_put(3S)
page	register PMfreezaddr	PMfreezaddr(3X)
PIR	register	PMoutpir(3N)
page	register and return an address to a	PMgetzaddr(3X)
PIR	register of a pipe DSP register of a pixel DSP	DEVpipe_get_pir(3S)
PIR	register of a pixel DSP	DEVpixel_get_pir(3S)
flag	registers /update overlay	DEVpixel overlay(3S)
page	registers for dynamic allocation	
page	registers used to access video and/	PMpagereg(3X)
o the	regular output FIFO PMfb on	
hireg	reserve DRAM and page registers for	PMzbrk(3X)
to a	reserved location in a pixel node	DEVpixel id write(3S)
reset	resets all FIFOs on a pipe board	DEVfifo_reset(3S)
		- • •

copy DRAM to vic PMcopyvtoz copy vic from a pipe feedback/ DEVfifo re DSP DEVpixel re memory of a pixel node DEVread PMz buffer PMgetz PIR register DEVpixel get n pipe DSP DEVpipe\_get\_n PMv0 PMv1 buffer PMget DEVget\_pixel, DEVget\_pix PMgets DSP's PIR register DEVpixel of a pipe DSP DEVpipe processor DEVpipe\_id\_p processor DEVpixel\_id\_p to the Pixel Machines pipelines a buffer PMqget qu frame buffer DEVget scan /PMgetcol, PMputrow, PMpu controller board/ DEVget\_color\_n DEVpipe get DSP DEVpixel\_get from a file DEVget\_image\_hea DSP DEVpipe r

values and/ DEVload color tak PMrdyoff turn links PMmsg\_exchange send PMclear f PMfreezaddr decrem PMclear fill a rectang a block of data to a pipe DSP's F of bytes from a pixel DSP's a message from a pixel DSP's initialize pixel board me set overlay mode in the pixel me block of data to a pixel DSP's P decrement references to a pa PMoutpir output a value to the section of/ PMgetzaddr load a p DEVpipe get pir read the DEVpixel get pir read the mode in all pixel processor's /PMset hireg reserve DRAM and pa /PMxlate macros to manipulate pa direct output commands to dynamic/ /PMset lowreg, PMset hi DSP's/ /write a node id block DEVfifo re

retrace PMvsync return an address to a section of/	
return its length	
return the value /color tables	
returns value /the color tables	
root function	
routines /server program for Pixel	oevprint(
row PMzaddr	PMzaodr(3)
safe 32-bit DRAM or VRAM copy	
scan lines from a frame buffer	
scancolumn from pixel memory/	PMgetrow(3)
scanline from a subscreen	PMgetscan(3)
scanline or scancolumn from pixel/	
scanline to a subscreen	
screen PMclear	PMclear(3)
screen space	
screen space	
screen space coordinate is in	
screen space coordinate is in	
screen space to processor space i	
screen space to processor space i/	
screen space to processor space j	
screen space (xmax) to processor	
screen space (xmin) to processor	
screen space (ymax) to processor	
screen space (ymin) to processor	
section of DRAM /load a page	
section of DRAM to another	
section of DRAM to another	
selected system commands	
selects the frame buffer to be	
semaphore	PMsetsem(3I
semaphore in the memory of one of	
DEV	
semaphore to clear	
send a block of data to a pixel	
send a message to the host that	
send a user message to the host	
send and receive data packet over	
serial DMA input pointer	
serial I/O	PMsioinit(3
serial I/O link direction	
serial I/O link direction	
	PMmsg exchange(3)
serial links PMmsg_exchange	
serial links PMmsg_exchange serial mode /configure server program for Pixel Machine	DEVfifo_serial(3 devprint(
serial links PMmsg_exchange	DEVfifo_serial(3 devprint(
serial links PMmsg_exchange serial mode /configure server program for Pixel Machine set of pipe nodes /load a Pixel	DEVfifo_serial(3 devprint( DEVpipe_boot(3
serial links PMmsg_exchange serial mode /configure server program for Pixel Machine set of pipe nodes /load a Pixel set of pixel nodes /load a Pixel	DEVfifo_serial(3 devprint( DEVpipe_boot(3
serial links PMmsg_exchange serial mode /configure server program for Pixel Machine set of pipe nodes /load a Pixel set of pixel nodes /load a Pixel set overlay mode in the pixel mode	DEVfifo_serial(3 devprint( DEVpipe_boot(3 DEVpixel_boot(3
serial links PMmsg_exchange serial mode /configure server program for Pixel Machine set of pipe nodes /load a Pixel set of pixel nodes /load a Pixel	DEVfifo_serial(3: devprint( DEVpipe_boot(3: DEVpixel_boot(3: DEVpixel_boot(3: 

DEVtools Reference Manual

PMgetzaddr load a page register and PMnorm normalize a 3D vector and from video controller board and from video controller board and PMsart saare Machine code that uses the print generate a ZRAM pointer to a PMcopy s DEVget scan line read one or more /read or write a scanline or PMgetscan read a /PMputrow, PMputcol read or write a PMputscan write a fill a rectangular region of the PMxat map subscreen coordinates to PMyat map subscreen coordinates to processor/ PMmyx test if a given processor/ PMmyy test if a given /map a linear function of x from /a linear function of x and y from /map a linear function of y from space (ihi) PMihi map from space (ilo) PMilo map from space (jhi) PMjhi map from space (jlo) PMjlo map from register and return an address to a PMcopyztoz copy from one PMqcopyztoz copy from one PMenable enable processing of displayed DEVpixel buffer PMsetsem set the the DSP/ /clear the software

synchronize and wait for vertical

PMwaitsem wait for DSP's PDR register DEVpixel\_put signals the completion/ PMhost\_exit PMusermsg serial links PMmsg\_exchange PMmsg\_setup set PMsioinit initialize DEVserial\_direction updates the PMsiodir set send and receive data packet over a pipe board to operate in code that uses the/ devprint a host Machine executable into specified Machine executable into specified register DEVpixel\_mode\_overlay

> PMmsg\_setup PMsiodir

em	set the semaphore	PMsetsem(3N)
ind	sets color lookup tables /file	DEVload color tables(3S)
om	shadow tables /turns off updating	
om	shadow tables /turns on updating	
ost	short integer /convert between	
32	short integer and host short/	
ost	short integers /convert	
32	short integers to host short/	
; to	signal completion, then call	
ady	signal off	
hat	signals the completion of a Pixel/	PMhost exit(3N)
the	software semaphore in the memory of	,
	DEVre	lease pipe semaphore(3H)
ght	sources PMIdot	
sor	space PMmyx test if a given screen	PMmyx(3X)
sor	space PMmyy test if a given screen	
en	space PMxat	
en	space PMyat	
en	space coordinate is in processor	
en	space coordinate is in processor	
sor	space i /map a linear function	
sor	space i and j /function of x and	
sor	space (ihi) PMihi map from	
sor	space (ilo) PMilo map from	
sor	space j /map a linear function	
sor	space (jhi) PMjhi map from	
sor	space (jlo) PMjlo map from	
en	space to processor space i /map	
en	space to processor space i and j	
en	space to processor space j /map	
en	space (xmax) to processor space	
en	space (xmin) to processor space	
en	space (ymax) to processor space	
en	space (ymin) to processor space	PMilo(3X)
dot	specialized dot product for light	PMldot(3M)
оa	specific pixel	PMpixaddr(3X)
nto	specified pipe nodes /begin	
into	specified pixel nodes /begin	DEVpixel run(3S)
nto	specified set of pipe nodes /load	
into	specified set of pixel nodes /load	
ora	specified time	PMdelav(3N)
and	specify functions to be called	DEVuser msg enable(3H)
sqrt	sqare root function	
on	standard error DEVerror	
eck	status of node's ID	
eck	status of node's ID	DEVpixel id check(3S)
da	stream of bytes from a pixel DSP's	DEVpixel get(3S)
da	stream of bytes from the PIR of a	
lata	structure used for FIFO commands	
lata	structure used for FIFO commands	
na	subscreen	
oa	subscreen	<b>U</b> ( )

PMsetsem of gamma calibration values and of color lookup tables from DSP32 short integer and host DEVbswaps convert between DSP32 from DSP32 short integers to host DEVswap\_short convert from DSP32 DEVexit /wait for pixel nodes to PMrdyoff turn the ready /send a message to the host that one of the DSP/ /clear the

specialized dot product for lig space coordinate is in process space coordinate is in process map subscreen coordinates to scree map subscreen coordinates to scree space PMmyx test if a given scree space PMmyy test if a given scree of x from screen space to process y from screen space to process screen space (xmax) to process screen space (xmin) to process of y from screen space to process screen space (ymax) to process screen space (ymin) to process a linear function of x from scree /function of x and y from scree a linear function of y from scree (ihi) PMihi map from scree

- (ilo) PMilo map from screen
- (jhi) PMjhi map from screen
- (jlo) PMjlo map from scree sources PMId

PMpixaddr generate a pointer to a execution of programs loaded into execution of programs loaded into a Pixel Machine executable into a Pixel Machine executable into PMdelay do nothing for a /define a message code and PMsqrt generate an error message on

DEVpipe\_id\_check check DEVpixel\_id\_check check PIR register\_DEVpixel\_get read a pipe DSP\_DEVpipe\_get read a PMcommand data PMcommand data PMgetscan read a scanline from a PMputscan write a scanline to a

subscreen coordinates to screen ...... PMxat(3X) subscreen coordinates to screen ...... PMyat(3X) subscreens ..... PMapply(3X) subscreens /scanline or scancolumn ...... PMgetrow(3X) swap front and back pixel buffers ..... PMswapbuff(3X) swap meaning of back buffer ..... PMswapback(3) switch primary and alternate pipes ..... DEVswap pipe(3H) synchronize PMpsync ..... PMpsync(3X) synchronize and wait for vertical ..... PMvsync(3X) system DEVswap\_pipe switch primary ..... DEVswap\_pipe(3H) system commands PMenable ..... PMenable(3N) tables /file of gamma calibration ...... DEVload color tables(3S) tables /turns off updating ..... DEVshadow off(3S) tables /turns on updating ...... DEVshadow\_on(3S) tables from shadow tables ...... DEVshadow off(3S) tables from shadow tables ...... DEVshadow on(3S) tables from video controller board ..... DEVput color map(3S) tables from video controller board/ ..... DEVget color map(3S) test if a given screen space ...... PMmyx(3X) test if a given screen space ...... PMmyy(3X) trigonometric function ...... PMsin(3M) trigonometric function to compute ...... PMcos(3M) turn overlay on or off ..... PMoverlay(3P) turn the PM FLAG LED on or off ..... PMflagled(3X) turn the PM RDY LED on or off ..... PMrdyled(3X) turn the ready signal off ..... PMrdyoff(3X) turns off updating of color lookup ..... DEVshadow\_off(3S) turns on updating of color lookup ...... DEVshadow on(3S) until control of the broadcast bus ...... PMswap pipe(3H) update color tables from video ...... DEVput color map(3S) update overlay mode in all pixel ...... DEVpixel overlay(3S) updates the serial I/O link ...... DEVserial direction(3S) updating of color lookup tables ...... DEVshadow off(3S) updating of color lookup tables ..... DEVshadow on(3S) upload an image from a Pixel ..... devsave(1) user message to the host ...... PMusermsg(3N) user program /DEVopen system ..... DEVopen(3S) uses the print routines /server ...... devprint(1) using DEVtools devcc C ..... devcc(1) value /read the color tables from ...... DEVget color map(3S) value /color tables from video ...... DEVput color map(3S) value /macro that converts floating ...... PMfloat color(3N) value /macro that converts ..... PMint color(3N) value from the Z buffer ..... PMgetzbuf(3X) value to an integer PMcolor int ..... PMcolor int(3N) value to floating point number ..... PMcolor float(3N) value to internal color value ..... PMfloat color(3N) value to the PIR register ..... PMoutpir(3N) value to the Z buffer ..... PMputzbuf(3X) values and sets color lookup tables ..... DEVload\_color\_tables(3S) values from a pipe feedback FIFO ...... DEVfifo\_read(3S) values to a pipe FIFO ...... DEVfifo write(3S)

**DEVtools Reference Manual** 

space PMyat map PMapply apply a function to all from pixel memory without PMswapbuff **PMswapback** of a dual pipe system DEVswap pipe wait for all pixel processors to retrace PMvsync and alternate pipes of a dual pipe enable processing of selected values and sets color lookup of color lookup tables from shadow of color lookup tables from shadow /turns off updating of color lookup /turns on updating of color lookup and/ DEVput color map update color DEVget color map read the color coordinate is in processor/ PMmyx coordinate is in processor/ PMmyy PMsin

space PMxat map

the cosine of an angle PMcos PMoverlay

PMflagled PMrdyled PMrdyoff

tables from shadow/ DEVshadow off tables from shadow/ DEVshadow on is granted PMswap pipe wait controller board/ DEVput color map processor's flag/ DEVpixel overlay direction DEVserial direction from/ DEVshadow off turns off from shadow/ DEVshadow on turns on Machine to a file devsave PMusermsg send a make a Pixel machine available to a program for Pixel Machine code that compiler for Pixel Machine programs video controller board and returns controller board and return the point value to internal color an integer to an internal color PMgetzbuf read a float macro that converts internal color /macro that converts internal color /macro that converts floating point PMoutpir output a PMputzbuf write a float /reads file of gamma calibration /read a block of four byte /write a block of four byte

variable increments	
vector and return its length	
vertical retrace	
video and Z memory /to manipulate	
video controller board and return	
video controller board and returns	. DEVget color map(3S)
video RAM	(3X)
video RAM to DRAM	
VRAM	
VRAM copy	
wait for all pixel processors to	PMpsvnc(3X)
wait for pixel nodes to signal	DEVwait exit(3H)
wait for semaphore to clear	
wait for vertical retrace	
wait until control of the broadcast	PMswan pipe(3H)
without subscreens /a scanline	
write a block of data to a pipe	
write a block of four byte values	
write a buffer to a pipe DSP	
write a buffer to a pixel DSP	DEvpixel_write(3S)
write a float to the Z-buffer	
write a float value to the Z buffer	
write a node id block to a reserved	
write a Pixel Machine image header D	
write a pixel to buffer 0	PMv0put(3X)
write a pixel to buffer 1	
write a scanline or scancolumn from	
write a scanline to a subscreen	
write of a pixel to the current	PMqput(3X)
write opcode and parameter count to	PMputop(3P)
write opcode, parameter count, and	PMputcmd(3P)
write parameters to the output FIFO	
write pixels into the frame buffer	DEVput pixel(3S)
write to the Pixel Machines/	DEVwrite(3H)
writes a buffer of bytes into the Z	
x and y from screen space to/	
x from screen space to processor/	
(xmax) to processor space (ihi)	
(xmin) to processor space (ilo)	
y from screen space to processor/	
y from screen space to processor/	
(ymax) to processor space (jhi)	
(ymin) to processor space (jiii)	
Z huffer DMaetzhuf	
Z buffer PMgetzbuf	
Z buffer PMputzbuf	
z buffer	
Z memory /macros to manipulate page	PMpagereg(3X)
Z memory of a pixel node DEVread_z	
Z memory of a pixel node	
Z-buffer	
ZRAM pointer to a column	
ZRAM pointer to a row	PMzaddr(3X)

λ.

PMcopy v 32-bit copy with PMnorm normalize a 3D PMvsvnc synchronize and wait for page registers used to access the value /update color tables from value /read the color tables from copy DRAM to PMcopyvtoz copy PMcopyvtov copy blocks of PMcopy s safe 32-bit DRAM or synchronize PMpsync completion, then call/ DEVwait exit PMwaitsem PMvsync synchronize and bus is granted PMswap pipe or scancolumn from pixel memory DSP's PDR register DEVpipe put to a pipe FIFO DEVfifo write DEVpipe\_write DEVpixel write PMzput PMputzbuf location in a/ DEVpixel\_id\_write to a file DEVput\_image\_header PMv0put PMv1put pixel/ /PMputrow, PMputcol read or PMputscan buffer PMqput quick the output FIFO of a pipe/ PMputop parameters to the output/ PMputcmd of a pipe node PMputdata DEVput pixel, DEVput pixels /DEVreadn, DEVreadn alt, macros to memory of a pixel node DEVwrite z PMfxytoij map a linear function of PMfxtoi map a linear function of PMihi map from screen space PMilo map from screen space /map a linear function of x and PMfytoj map a linear function of PMjhi map from screen space PMjlo map from screen space read a float value from the write a float value to the PMzget read a float from the registers used to access video and read a buffer of bytes from the /writes a buffer of bytes into the PMzput write a float to the PMzaddrcol generate a PMzaddr generate a

d3as - DSP32 assembler

# SYNOPSIS

d3as [options] source\_files...

# DESCRIPTION

Filenames ending with .s or .i are assumed to be DSP32 assembly source files. Each specified source file is assembled, and a corresponding object file is created with a .o suffix. The valid options are:

- V	Print the version number and exit.
-N	Produce DSP32 object code. (Default mode)
-Q	Produce DSP32C object code.
-C	Retain comments through preprocessor (useful only with -P).
-P	Preprocess the named files and store them in corresponding files with the .i suffix.
$-\mathbf{D}n$	Define $n$ to the preprocessor with value 1.
$-\mathbf{D}n=v$	Define $n$ , an identifier, to the preprocessor as if by #define and give it value $v$ .
-Un	Undefine $n$ by removing any initial definition of $n$ .
-Idir	The #include files whose names do not begin with $/$ ( $\land$ on <i>MS-DOS</i> ) should be searched for in <i>dir</i> , before looking in the directories on the standard list. Thus, #include files whose names are enclosed in " " are searched for first in the directory of the <i>filename</i> argument, then in directories named in $-I$ options, and last in directories on a standard list. For #include files whose names are enclosed in $<>$ , the directory of the <i>filename</i> argument is not searched.
-In	(Lower-case L). Produce listing of assembly file. The $n$ , if specified is the page length (default is 66 lines).
-Ifile	(Lower-case L). Produce listing of assembly file and store in <i>file</i> . If <i>file</i> is not specified, the source file names are used (with a .l extension).
- <b>n</b>	Generate parity bits for the DSP32 device. Note: This option has the opposite effect that it had in previous versions of the assembler.
$-\mathbf{W}$	Turn off warning messages.
- <b>F</b>	Treat certain programming violations as warnings, rather than fatal errors. See section 3.4 of the DSP32 and DSP32C Support Software Library DSP32 and DSP32C C Language Compiler
-A	Do not invoke the C preprocessor.
-p	Whenever possible, translate each goto statement to a pc relative goto statement ( <b>pcgoto</b> ). Note that is option does not translate call statements to pc relative call statements ( <b>pccall</b> ).
-o file	Place output object file in <i>file</i> .

# DIRECTIVE

The assembler supports the following directives:

.rsect section name

This assembler directive allows the user to set up a relocatable program section. The one argument to .rsect is a legal identifier enclosed in quotes which is the name of the section.

- .= The .= directive is followed by a constant expression. It sets the current section's location counter to the constant value that is on the right of the equal sign. The expression cannot be external.
- .align This directive is used to assure that an instruction or data occurs on a legal boundary. It is usually used when data space is allocated. The directive has one argument, and integer constant that is used to determine that correct alignment.
- .global Once an identifier is used, it is known from that point on in the file. Therefore, every identifier in a file must be unique. The identifiers are not known across file boundaries. The .global directive is followed by a list of identifiers, separated by commas, that are to be made known across file boundaries. The identifiers on the directive line must be defined in that file, but are then available to other files that are linked with it.
- .extern If an identifier is listed as external, it is defined and listed as global in another file, but is known throughout the local file. The .extern directive is followed by a list of identifiers, separated by commas.
- .list Turn on listing. For use with the -I flag.
- .nolist Turn off listing. For use with the -l flag.
- .page Skip to the top of a new page. For use with the -I flag.

#### EXAMPLES

The command

d3as test.s

will produce a file test.o which contains the relocatable object code produced by assembling test.s.

The command

d3as -l test.s

will produce an assembly listing written to the file test.l. This command also produces a relocatable object file, test.o.

#### NEW FEATURES

A new form for an unconditional branch instruction is supported for both the DSP32 and the DSP32C. This instruction is:

#### pcgoto label

The assembler will produce a pc-relative goto which can be dynamically relocated without affecting the branch. Presently, there is a restriction that the label must be within the same section as the **pcgoto** instruction using it and within the same file. These restrictions may be lifted at a later time.

#### SEE ALSO

DSP32 C Support Software Library User Manual DSP32 and DSP32C Support Software Library DSP32 and DSP32C C Language Compiler d3sim(1) d3cc(1)

d3ld(1)

NAME d3cc - DSP32 C language co	mpiler
SYNOPSIS d3cc options source_files	
DESCRIPTION	
The valid options are:	
-N	Produce DSP32 object code. (Default mode)
-Q	Produce DSP32C object code.
- <b>P</b>	Invoke the C preprocessor only. For each <i>file.c</i> , this generates a <i>file.p</i> containing the preprocessed C source code.
<b>-S</b>	Invoke the preprocessor and compiler only. This generates assembly source files (.s extension) from C source files.
— <b>i</b>	Invoke the compiler and optimizer only. This generates optimized assembly files (.i extension) from C or assembly source files (.c or .s extension, respectively).
- <b>c</b>	Invoke the compiler, optimizer, and assembler only. This generates object files (.o extension) from C or assembly source files (.c or .s extension, respectively).
-1	Generate a listing file (.l extension) of assembled files. The listing is useful for assembly-level debugging.
-t textseg	Causes the compiler to load all the program text in the compiled files in a section called <i>textseg</i> instead of the default section .text.
- <b>d</b> dataseg	Causes the compiler to load all the global and static data in the compiled files in a section called <i>dataseg</i> instead of the default section .data.
- <b>m</b> mapfile	Specifies an alternate memory configuration file (ifile) for use by the linker. The default ifiles are mem32.map (for the DSP32) and mem32c.map (for the DSP32C) in the directory \$DSP32SL/lib.
-s startfile	Specifies an alternate start-up file for use by the linker. The default start- up files are <b>crt0_32.0</b> (for the DSP32) and <b>crt0_32c.0</b> (for the DSP32C) in the directory \$DSP32SL/lib.
-o outfile	Specifies the name of the output file. The default output file is a.out.
-Lxx	Includes the library libxx32.a or libxx32c.a, depending on whether DSP32 or DSP32C code is being generated.
-Wc,arg1,[arg2]]	
	Passes the specified argument(s) $(arg1)$ to pass c, where c is one of $\{\mathbf{p}, \mathbf{c}, 0, \mathbf{a}, \text{ or } \mathbf{l}\}$ indicating the preprocessor, compiler, optimizer, assembler, or linker, respectively.
$-\mathbf{D}n$	Define $n$ to the preprocessor with value 1.
$-\mathbf{D}n=v$	Define $n$ , an identifier, to the preprocessor as if by #define and give it value $v$ .
$-\mathbf{U}n$	Undefine $n$ by removing any initial definition of $n$ .
- <b>I</b> dir	The #include files whose names do not begin with / (\ on MS-DOS) should be searched for in <i>dir</i> , before looking in the directories on the standard list. Thus, #include files whose names are enclosed in " " are searched for first in the directory of the <i>filename</i> argument, then in

( )

directories named in -I options, and last in directories on a standard list. For #include files whose names are enclosed in <>, the directory of the *filename* argument is not searched.

Generate parity bits for the DSP32 device. Note: This option has the opposite effect that it had in previous versions of the assembler.

-T Trace program execution. d3cc prints command lines used to invoke the preprocessor, compiler, optimizer, assembler, and linker. Useful for debugging problems with d3cc command strings.

#### SEE ALSO

-n

DSP32 C Language Compiler User Manual d3as(1) d3sim(1) d3ld(1)

d3ld - DSP32 link editor

### SYNOPSIS

d3ld [options] [ifile] obj\_files...

# DESCRIPTION

The d3ld command links the named *obj\_files* object files, produced by d3as or d3cc, and puts the resulting object file into a.out unless otherwise specified. The *ifile* is an ASCII file containing directives.

The valid options are:

-a	Produces an absolute, executable file; gives warnings for undefined references. Relocation information is stripped from the output file unless the $-r$ option is given. The $-r$ option is needed only when an absolute file should retain its relocation information (not the normal case). If neither $-a$ nor $-r$ is given, $-a$ is assumed.
-f fill	Sets the default fill pattern for "holes" within an output section as well as initialized <i>bss</i> sections. The argument <i>fill</i> is a two-byte constant.
-lx	Searches a library libx.a, where x is up to nine characters. A library is searched when its name is encountered, so the placement of a $-1$ is significant. By default, libraries are located in the directory lib within the directory specified by the environment variable DSP32SL.
- <b>m</b>	Produces a map or listing of the input/output sections (including holes) on the standard output.
-o outfile	Produces an output object file by the name <i>outfile</i> . The default name of the object file is <b>a.out</b> .
- <b>r</b>	Retains relocation entries in the output object file. Relocation entries must be saved if the output file is to become an input file in a subsequent <b>Id</b> run. The link editor does not complain about unresolved references, and the output file is not executed.
-s	Strips line number entries and symbol table information from the output object file. This function can also be performed using the utility <b>d3strip</b> .
– u symname	Enters <i>symname</i> as an undefined symbol in the symbol table. This is useful for loading entirely from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine.
- <b>x</b>	Does not preserve local (nonglobal) symbols in the output symbol table; enters external and static symbols only. This option saves some space in the output file.
-L dir	Changes the algorithm of searching for libx.a to look in <i>dir</i> before looking in DSP32SL/lib. This option is effective only if it precedes the $-1$ option on the command line.
- <b>M</b>	Outputs a message for each multiply-defined external definition. How- ever, if the objects loaded include debugging information, extraneous out- put is produced.
-N	Puts the data section immediately following the text in the output file.
$-\mathbf{V}$	Outputs a message giving information about the version of d3ld being used.
-n	Generate parity bits for DSP32 device. Note: This option has the

# opposite effect that it had in previous versions of the linker.

Flags can be combined with file names on both the command line and in an *ifile*. The ordering of flags is insignificant with the exception of the -I flag for specifying libraries. Libraries are searched as they are encountered for any undefined external references.

SEE ALSO

DSP32 C Support Software Library User Manual d3as(1) d3sim(1) d3cc(1)

NAME	
d3sim - DSP32 link editor	
SYNOPSIS d3sim [options] file	
DESCRIPTION The file is the DSP32 execu	table program file that is being simulated. The valid options are:
- <b>c</b>	The -c option must be used with programs that were compiled by the DSP32/DSP32C C compiler. This option allows breakpoints to operate correctly, allows the <b>printf</b> function to be used in the program, and also allows registers r14, r18, and r19 to be referred to as <b>sp</b> (stack pointer), <b>rp</b> (return pointer), and <b>ir</b> (increment register), respectively. The <b>ftrace</b> command (trace function calls) is also available.
- <b>T</b>	Time run. The time taken to run the simulation on the host computer is displayed (in seconds) at the end of a simulation run. (This is not the time that the physical device takes to run the same program).
- <b>e</b>	Exec file. This option causes a text file of commands to be executed. The name of the file is assumed to be file.ex, where file is the name of the program file. It is useful to produce such a file and put into it definitions of any functions that would be useful in debugging that particular program.
$-\mathbf{m}n$	Memory Mode Specification. This option specifies the memory mode. The value of $n$ can be 0 through 3 for DSP32 programs or 0 through 7 for DSP32C programs. If it is not specified, the mode defaults to 2 for the DSP32 or 6 for the DSP32C. If no mode flag is present, d3sim allows writes to ROM with no complaint.
-1	Log Commands. This option causes "command logging" to be turned on. If input is from a terminal, it gets written to a file called log.cmd. If input is from a file, then as it is read, it gets written to standard output. Thus, each line of output is preceded by the command line that caused it, which can be useful in identifying unexpected results.
- b	Turn off breakpoint verbose mode. No message is printed when a break- point occurs. Note: This option has the opposite effect that it had in pre- vious versions of the simulator.
- <b>d</b> #	Development system specifier. Sets the simulator in development system mode. The # specifies the number of the DSP32 development system that is being controlled (see the WE®DSP32-DS Development System User Manual for details).
– D/dev/alt_port	Device Driver Select. UNIX SYSTEM ONLY The /dev/alt-port is used if the DSP32 development system is connected to a port other than the user's tty port. The /dev/alt_port is the UNIX System device driver of the port to which the development system is con- nected (see the WE®DSP32-DS Digital Signal Processor Development System User Manual for details).
-C file.cfg	DSP32C Development System in use. Refer to the WE®DSP32C Development System User Manual for complete details.
-Z	Disables "dirty-zero" checking in the DAU. By default, a dirty-zero error occurs when a number has a nonzero exponent and a mantissa of zero.

- <b>p</b>	Turn on profiling. The profiling feature requires a large amount of memory, which may cause problems on some systems. Therefore, it is not active by default.
- <b>R</b>	DSP32C mode only. Run DSP32C as a ROM device.
-P	Set default pcw value to 0. Normal default is 0x3f.
-An	DSP32C mode only. Set number of wait states for external memory bank A to n.
-Bn	DSP32C mode only. Set number of wait states for external memory bank B to $n$ .
$-\mathbf{w} n$	Sets the number of conflict wait states to remember to $n$ . This determines the number of conflict wait states that are displayed when the waits command is issued.
-S #	Enable stack range checking. The argument # is an upper bound the stack pointer should not exceed.

Prior to accepting user command input, d3sim loads memory from the given *file* and initializes as if a chip reset has occured.

# SEE ALSO

DSP32 C Support Software Library User Manual d3as(1) d3cc(1) d3ld(1)

devcc – C compiler for Pixel Machine programs using DEVtools

#### SYNOPSIS

devcc <d3cc options> [-pixel | -pipe] <source files>

#### DESCRIPTION

devcc is the DSP32 compiler used with DEVtools programs. It is the same as d3cc but it knows about Pixel Machine specific files. In addition to the directories searched by d3cc for include files, devcc also searches \$HYPER\_PATH/devtools/include. devcc also passes the correct startup file and loader directive file (*ifile*) to d3ld as well as the \$HYPER PATH/devtools/lib/libpm.a library.

devcc takes all the options that d3cc does plus -pixel (default) and -pipe options.

The -pipe option is used to link pipe programs and causes crt0 pipe.o and pipe ifile to be used.

The -pixel option is for pixel programs and causes crt0 pixel.o and pixel ifile to be used.

#### NOTES

If users want to use **printf** with **d3sim**, they should include **\$D\$P32SL/include/printf.c** on the **devcc** command line to prevent loading the **printf** that is included in **libpm.a**.

### SEE ALSO

DSP32 C Language Compiler User Manual

DSP32 Support Software Library and DSP32 C Language Compiler Version 1.3.1 Addendum

devdisp – download an image from a file to a Pixel Machine.

#### **SYNOPSIS**

```
devdisp [-p initx inity] [-s npixels nlines] [-o xoffset yoffset] [-b buffer] [-d] [-v] [-u] file
```

# DESCRIPTION

devdisp is used to download an image from a file to a Pixel Machine. The *file* specified must be in DEVtools image format as specified in DEVimage\_header(4).

The following options are supported:

- <b>p</b> initx inity	the image download will begin at pixel ( <i>initx</i> , <i>inity</i> ). Default is $(0,0)$ (upper left hand corner of the screen).
-s npixels nlines	a rectangular section of pixels specified by ( <i>npixels, nlines</i> ) will be down-loaded. Default is the size of the image as specified in the file.
– o xoffset yoffset	if specified, <i>xoffset</i> pixels and <i>yoffset</i> lines are skipped in the image file before downloading. This is used to download only a portion of the image file. Default is 0, 0.
-d	the image download will begin at the pixel specified in the image file. This option is useful when an image was saved from a specific location on the screen and the user wishes to display it at the same location. This option overrides the $-\mathbf{p}$ option.
- <b>b</b> buffer	the image will be downloaded to the specified portion of the frame buffer. Valid values for <b>buffer</b> are:
	<i>front</i> – pixels are downloaded to the front (currently displayed) buffer (default).
	<i>back</i> – pixels are downloaded to the back (currently non-displayed) buffer.
	vram0 – pixels are downloaded to VRAM0.
	vram1 – pixels are downloaded to VRAM1.
	zram – pixels are downloaded to ZRAM.
$-\mathbf{v}$	verbose output will be written to the standard output.
-u NS	print usage information.
	on success, non-zero on failure.

### NOTES

RETURNS

devdisp downloads code into the pipe and pixel nodes to perform the image download, consequently any programs that had been downloaded will be overwritten.

SEE ALSO

DEVimage\_header(4) DEVput\_scan\_line(3H) devsave(1) picdisp(1) in the PIClib *Reference Manual* raydisp(1) in the RAYlib *Reference Manual* 

devprint – a host server program for Pixel Machine code that uses the print routines

#### SYNOPSIS

devprint [-d node all] [-g node all] [-u] [-i] [-n]

#### DESCRIPTION

devprint is a program that runs on the host system that polls a selected set of pipe and/or pixel nodes and performs the host processing required by any system messages sent from the nodes, usually the messages for PMhost exit, PMsiodir and printf.

The following options may be used:

-d node poll pixel node node for print messages

-dall poll all pixel nodes for print messages

-g node

poll pipe node node for print messages

- -gall poll all pipe nodes for print messsages
- -i print node identification information for node **printf** commands
- -n causes devprint to poll all nodes specified, but discards all the messages except from the first pipe and pixel node specified on the command line. This is used for debugging when it is not necessary to see the output of all of the nodes, but they must be polled so they do not hang waiting for the host to read a message. Care must be taken when using this option because commands executed on the other nodes will not function properly if they expect a response from the host.
- -u print command usage format

If no node specification is provided, all pipe and pixel nodes are polled for print messages.

#### **EXAMPLES**

devprint - poll and print for all nodes

devprint -n - only prints output for Pipe and Pixel node 0

devprint -gall - dall - n - only prints output for Pipe and Pixel node 0

devprint -n - g8 - d5 - prints for pipe #8 and Pixel #5

# SEE ALSO

DEVpoll\_nodes(3S) PMhost\_exit(3N) PMsiodir(3X) printf(3N)

devsave - upload an image from a Pixel Machine to a file

# SYNOPSIS

```
devsave [-p initx inity] [-s npixels nlines] [-b buffer] [-m mode] [-v] [-u] file
```

# DESCRIPTION

**devsave** is used to upload an image that exists in the frame buffer of a Pixel Machine into a file on the host computer. The image is stored in *file* in the format specified by *mode*. The uploaded file will contain an initial DEVtools header. See the DEVimage\_header(4) manual page for a description of the Pixel Machines image header. Each pixel component (red, green, blue and alpha) consumes 8 bits and is byte aligned. *file* will be overwritten if it exists.

The following options are supported:

- <b>p</b> initx inity	the image upload will begin at pixel ( <i>initx</i> , <i>inity</i> ). Default is (0,0) (upper left hand corner of the screen).
-s npixels nlines	a rectangular section of pixels specified by (npixels, nlines) will be uploaded. Default is the size of the screen.
- <b>b</b> buffer	the image will be uploaded from the specified portion of the frame buffer. Valid values for <i>buffer</i> are:
	front – pixels are uploaded from the front (currently displayed) buffer (default).
	<i>back</i> – pixels are uploaded from the back (currently non-displayed) buffer.
	vram0 - pixels are uploaded from VRAM0.
	vraml – pixels are uploaded from VRAM1.
	zram – pixels are uploaded from ZRAM.
- <b>m</b> mode	the image will be uploaded according to the format specified in <i>mode</i> . Valid values for <i>mode</i> are:
	rgba - pixels are stored in red, green, blue, alpha format (default).
	rgb – pixels are stored in red, green, blue format.
	a – only the alpha component of the pixel is stored.
	b – only the blue component of the pixel is stored.
	g – only the green component of the pixel is stored.
	r – only the red component of the pixel is stored.
	$a\_mono$ – the alpha component of the pixel is stored, and the image header is set to mono (for later monochrome display).
	$b\_mono$ – the blue component of the pixel is stored, and the image header is set to mono (for later monochrome display).
	$g\_mono$ – the green component of the pixel is stored, and the image header is set to mono (for later monochrome display).
	$r_{mono}$ – the red component of the pixel is stored, and the image header is set to mono (for later monochrome display).
	mono - pixels will be read 8 bits at a time from ZRAM only.

16 - pixels will be read 16 bits at a time from ZRAM only.

dsp - pixels will be read 32 bits at a time from ZRAM only.

ieee - DSP floats will be converted to IEEE floats in ZRAM and uploaded. After the floats are uploaded, the values in ZRAM are converted back to DSP floats.

verbose output will be written to the standard output. print usage information.

#### RETURNS

 $-\mathbf{v}$ 

- u

The exit code will be 0 upon success, non-zero on failure.

NOTES

devsave downloads code into the pipe and pixel nodes to perform the image upload, consequently any programs that had been downloaded will be overwritten.

SEE ALSO

DEVimage\_header(4) DEVget\_scan\_line(3H) devdisp(1) picsave(1) in the PIClib *Reference Manual* raysave(1) in the RAYlib *Reference Manual* 

DEVbswapl - convert between DSP32 long integer and host long integer

### SYNOPSIS

#include <host/devtools.h>

long DEVbswapl(number)
long number;

# DESCRIPTION

**DEVbswapl** (byte swap long) converts a long integer in DSP32 format to a long integer in the host format, and vice-versa. **DEVbswapl** is implemented as a macro which returns the value of *number* with the bytes in reverse order.

#### SEE ALSO

DEVswap\_long(3S) DEVbswapl(3S)

DEVbswaps - convert between DSP32 short integer and host short integer

# SYNOPSIS

#include <host/devtools.h>

short DEVbswaps(number)
short number;

## DESCRIPTION

**DEVbswaps** (byte swap short) converts a short integer in DSP32 format to short integers in the host format and vice-versa. **DEVbswaps** is implemented as a macro which returns the value of *number* with the high and low bytes swapped.

#### SEE ALSO

DEVswap\_short(3S) DEVbswaps(3S)
**DEVclose** – closes the Pixel Machine

## SYNOPSIS

void DEVclose()

# DESCRIPTION

**DEVclose** closes the Pixel Machine designated by the environment variable HYPER\_UNIT. Closing the device consists of closing the file associated with the VME device, releasing the memory blocks that were mapped to the device, and removing the lock file.

The system status file is updated to reflect any changes that may have occurred during the execution of the program since the device was opened.

# NOTES

**DEVexit** rather than **DEVclose** is usually used. **DEVopen** and **DEVclose** are provided for users that require lower level control of the system.

SEE ALSO

DEVexit(3H) DEVopen(3S)

**DEVdsp\_ieee** – convert from the DSP32 floating–point format to the IEEE floating–point format

## SYNOPSIS

#include <host/devtools.h>
float DEVdsp\_ieee(n)
long n;

## DESCRIPTION

The host and the DSP32 use different formats for floating point numbers. DEVdsp\_ieee converts a single 32 bit floating point number in DSP32 format to the IEEE floating point format used by the host. The number to be converted is stored in the 32 bit long n. The contents of n must be in the correct host byte order. A value read from the Pixel Machine must be converted using DEVbswapl() or DEVswap long() before calling DEVdsp ieee().

## RETURNS

DEVdsp\_ieee returns a floating point number with the same value as the DSP32 floating point number.

### SEE ALSO

DEVieee\_dsp(3S) DEVbswapl(3S) DEVswap\_short(3S) DEVswap\_long(3S)

DEVerror - generate an error message on standard error

### **SYNOPSIS**

#include <host/devtools.h>
#include <host/deverror.h>

void DEVerror(msg)
char \*msg;

char DEVerror\_msg[];
int DEVerrno;

### DESCRIPTION

**DEVerror** is the DEVtools equivalent of the UNIX system **perror**() function. It is used to generate an error message on standard error describing the last error that occurred during a call to a DEVtools host function.

A message of the form:

*msg*: error message

is generated.

Error messages can also be formatted by user programs by accessing the global variable DEVerror\_msg. User programs can check for specific errors by comparing the global variable DEVerrno with the symbolic names defined in the deverror.h include file.

## NOTES

It is possible for some DEVtools routines to fail because of errors returned from system calls. When this occurs, DEVerrno contains the value DEV\_ERR\_SYSTEMERR, and the contents of DEVerror\_msg is undefined. Therefore, user error message handlers should not display DEVerror msg for system errors.

#### SEE ALSO

perror on host system

DEVexit - halts processors, closes Pixel Machine device

# SYNOPSIS

void DEVexit()

# DESCRIPTION

**DEVexit** halts the processors, closes the device associated with the Pixel Machine, and restores the default handling of signals intercepted by **DEVinit**. **DEVexit** should always be called before exiting any host program that uses **DEVinit**.

## NOTES

**DEVexit** does not wait for the Pixel Machine to finish any outstanding commands. Use **DEVwait\_exit** to guarantee that the pixel nodes are done.

SEE ALSO

DEVclose(3S) DEVinit(3H) DEVwait\_exit(3H)

**DEVfifo\_parallel** – configure a pipe board to operate in parallel mode

# SYNOPSIS

#include <host/devtools.h>

void DEVfifo\_parallel(system, fifo)
DEVpixel\_system \*system;
int fifo;

# DESCRIPTION

**DEVfifo\_parallel** configures a pipe board to operate in parallel mode. This mode can only be used in systems with two pipe boards. A call to this function must be made for each pipe card in the system. *fifo* is the number of the pipe board whose FIFO is to be configured in parallel.

### NOTES

**DEVfifo\_parallel** is automatically called by **DEVinit** and **DEVopen** on dual parallel pipe systems as specified by the HYPER MODEL and HYPER PIPE environment variables.

## SEE ALSO

DEVfifo\_serial(3S) DEVinit(3H) DEVopen(3S)

DEVfifo read - read a block of four byte values from a pipe feedback FIFO

# SYNOPSIS

#include <host/devtools.h>

int DEVfifo\_read(input, input\_flags, buffer, nwords)
DEVulong \*input;
DEVbyte \*input\_flags;
DEVulong \*buffer;
int nwords;

## DESCRIPTION

**DEVfifo\_read** reads a block of four byte values from a pipe node feedback FIFO. This is done by copying the data from the memory mapped address of the feedback FIFO.

*input* is a pointer to the memory mapped area that the data is to be read from. *input\_flags* is a pointer to the memory mapped location of the input flags of the pipe board.

buffer is a pointer to the location into which the data is to be read. *nwords* is the number of four byte values to be read.

## NOTES

THE DEVcread macros should be used for most applications.

DEVfifo\_read always returns zero.

**DEVfifo read** cannot be used on a system without pipe boards.

### SEE ALSO

DEVfifo\_write(3S) DEVwrite(3H)

DEVfifo\_reset - resets all FIFOs on a pipe board

# SYNOPSIS

#include <host/devtools.h>

void DEVfifo\_reset(pixel\_system,fifo)
DEVpixel\_system \*pixel\_system;
int fifo;

# DESCRIPTION

DEVfifo\_reset resets all the FIFOs on a pipe board. fifo is the number of the pipe board to be reset.

# NOTES

Resetting a pipe board empties all of its FIFOs.

DEVfifo\_reset is automatically called by DEVinit and DEVopen.

1

**DEVfifo\_serial** – configure a pipe board to operate in serial mode

# SYNOPSIS

#include <host/devtools.h>

void DEVfifo\_serial(pixel\_system,fifo)
DEVpixel\_system \*pixel\_system;
int fifo;

# DESCRIPTION

**DEVfifo\_serial** configures a pipe board to operate in serial mode. This mode can be used in systems with two pipe boards. A call must be made for each pipe card in the system. *fifo* is the number of the pipe card.

# NOTES

**DEVfifo\_serial** is automatically called by **DEVinit** and **DEVopen** on dual serial pipe systems as specified by the HYPER\_MODEL and HYPER\_PIPE environment variables.

## SEE ALSO

DEVfifo\_parallel(3S)

**DEVfifo\_write** – write a block of four byte values to a pipe FIFO

# SYNOPSIS

#include <host/devtools.h>

int DEVfifo\_write(output, output\_flags, buffer, nwords)
DEVulong \*output;
DEVbyte \*output\_flags;
DEVulong \*buffer;
int nwords;

### DESCRIPTION

**DEVfifo\_write** writes a block of four byte values to a pipe FIFO. This is done by copying the data to the memory mapped address of the FIFO.

*output* is a pointer to the memory mapped area that the data is to be written to. *output\_flags* is a pointer to the memory mapped location of the output flags of the pipe board.

buffer is a pointer to the data to be written. nwords is the number of four byte values to be written.

# NOTES

DEVfifo\_write always returns zero.

DEVfifo\_write cannot be used to write directly to the broadcast bus FIFO.

The DEVwrite macros provide a more efficient mechanism to write to a pipe FIFO.

SEE ALSO

DEVfifo\_read(3S) DEVwrite(3H)

DEVget\_color\_map - read the color tables from video controller board and returns value

# SYNOPSIS

#include <host/devtools.h>

void DEV	et_color_map(pixel_system, r, g, b)	
DEVpixel	system *pixel_system;	
int	r[DEV_VIDEO_TABLE];	
int	g[DEV_VIDEO_TABLE];	
int	b[DEV_VIDEO_TABLE];	

# DESCRIPTION

**DEVget\_color\_map** reads the color tables from the video controller board and returns the values to the caller. Each color table contains 256 entries; each entry is a 10-bit value (0-1023).

### SEE ALSO

DEVput\_color\_map(3S)

DEVget image header - read the Pixel Machine image header from a file

### **SYNOPSIS**

#include <stdio.h>
#include <host/devtools.h>
#include <host/devimage.h>
#include <host/deverror.h>

int DEVget\_image\_header(file, image\_header, optional\_header) FILE \*file; DEVimage\_header \*image\_header; DEVbyte \*\*optional header;

#### DESCRIPTION

**DEVget\_image\_header** reads the DEVimage\_header and the optional header (if one exists) from the specified file and returns them to the caller.

*file* is a file descriptor obtained from a previous call to *fopen(3)*. The file must have been successfully opened for reading and the file pointer should be pointing to the beginning of the file (i.e., no previous reads have been issued). Upon return from **DEVget\_image\_header**, the file pointer will be set to the beginning of the pixel data (i.e., past the image and optional headers).

**DEVget\_image\_header** will read in the first DEV\_IMAGE\_HEADER\_SIZE bytes from the file, convert them from ASCII into unsigned longs and place them into the correct locations in the structure pointed to by *image\_header*. Except for the *magic* and *optional\_header\_size* fields, none of the information in the header is checked for validity.

If an optional header is present (image\_header->optional\_header\_size is not 0), memory will be allocated (via *malloc(3)*) and *image\_header->optional\_header\_size* bytes will be read. A pointer to the allocated memory will be returned in *\*optional\_header*. If no optional header is present, *\*optional header* will be set to NULL.

#### RETURNS

**DEVget\_image\_header** returns 0 upon success and -1 on failure. **DEVget\_image\_header** will set **DEVerrno** to indicate the reason for failure:

DEV ERR BAD MAGIC: the magic number is not DEV IMAGE MAGIC.

DEV\_ERR\_READ\_ERR: an error was returned by the *fread(3)* system call while reading either the image header or the optional header.

### SEE ALSO

DEVimage\_header(4) DEVput\_image\_header(3S)

**DEVget\_pixel**, **DEVget\_pixels** – read a pixel from the frame buffer

## **SYNOPSIS**

#include <host/devtools.h>

void DEVget\_pixel(system, buffer, x, y, r, g, b, o)
DEVpixel\_system \*system;
int buffer, x, y;
short \*r, \*g, \*b, \*o;

void DEVget\_pixels(system, buffer, x, y, r, g, b, o, npixl)
DEVpixel\_system \*system;
int buffer, x, y;
short \*r, \*g, \*b, \*o;
int npixl;

#### DESCRIPTION

**DEVget\_pixel** reads a pixel from the frame buffer. By using this routine, a program can read the Pixel Machine frame buffer without having to deal with the details of how the frame memory is organized on different models of the system.

system is a pointer to the system description information returned by **DEVinit**. buffer indicates which frame buffer is to be updated (must be the value 0 or 1). x is the x coordinate, y is the y coordinate. r, g, b, and o are pointers to the locations into which the values of the red, green, blue, and overlay values from the frame buffer are to be stored.

**DEVget\_pixels** reads a sequence of pixels for a single scan line. npixl is the number of pixels to be read. r, g, b, and o point to the locations into which the values of red, green, blue, and overlay values to be stored.

# NOTES

DEVget\_scan\_line provides a more efficient and versatile way to upload images.

### SEE ALSO

DEVpixel\_read(3S) DEVget\_scan\_line(3H) DEVinit(3H)

DEVget scan line - read one or more scan lines from a frame buffer

#### **SYNOPSIS**

#include <host/devtools.h>
#include <host/devimage.h>

int DEVget\_scan\_line(system, x, y, npixl, nlines, mode, pixels)
DEVpixel\_system \*system;
unsigned int x, y;
unsigned int npixl, nlines, mode;
DEVbyte \*pixels;

## DESCRIPTION

**DEVget\_scan\_line** reads one or more scan lines from the frame buffer and packs the pixels into *pixels* according to the mode specified by *mode*. By using this routine, a program can read scan lines from a Pixel Machine frame buffer without having to deal with the details of how the frame memory is organized on different models of the system.

system is a pointer to the system description information returned by **DEVinit**. x is the starting x screen coordinate, y is the starting y screen coordinate.

**DEVget\_scan\_line** reads a sequence of pixels for one or more scan lines. *npixl* is the number of pixels to be read from each scan line, *nlines* scan lines will be read. *pixels* points to the location into which the pixel values will be stored.

The buffer pointed to by *pixels* must be large enough to store (*npixl* \* *nlines* \* pixel size) bytes, where the pixel size is determined by the *mode* argument as described below. In all cases, pixels will be stored in *pixels* in the following order: (x,y), (x+1,y), ..., (x+npixl-1,y), (x,y+1), ..., (x+npixl-1,y+nlines-1).

The *mode* argument is used to specify two independent pieces of information: how the pixels will be stored in the array pointed to by *pixels*, and which portion of Pixel Machine memory the data should be copied from. These two values are or'ed into the *mode* argument. Valid values for *mode* and their results are:

DEV\_RGBA\_PACKED\_PIXELS: pixels will be stored in *pixels*, 4 bytes to a pixel, in the following order: red, green, blue, alpha.

DEV\_RGB\_PACKED\_PIXELS: pixels will be stored in *pixels*, 3 bytes to a pixel, in the following order: red, green, blue.

DEV\_MONO\_R\_PIXELS: pixels will be stored in *pixels*, 1 byte to a pixel, with the red component of the pixel actually being stored.

DEV\_MONO\_G\_PIXELS: pixels will be stored in *pixels*, 1 byte to a pixel, with the green component of the pixel actually being stored.

DEV\_MONO\_B\_PIXELS: pixels will be stored in *pixels*, 1 byte to a pixel, with the blue component of the pixel actually being stored.

DEV\_MONO\_A\_PIXELS: pixels will be stored in *pixels*, 1 byte to a pixel, with the alpha (overlay) component of the pixel actually being stored.

DEV\_MONO\_PIXELS: pixels will be stored in *pixels*, 1 byte to a pixel. This option is only available when reading from DEV ZRAM BUFFER.

DEV\_MONO\_16\_PIXELS: pixels will be stored in *pixels*, 2 bytes to a pixel. This option is only available when reading from DEV\_ZRAM\_BUFFER.

1

DEV\_DSP\_FLOAT\_PIXELS: pixels will be stored in *pixels*, 4 bytes to a pixel. This option is only available when reading from DEV ZRAM BUFFER.

DEV\_IEEE\_FLOAT\_PIXELS: DSP floating point values in ZRAM will be converted to IEEE floating point pixels in ZRAM, then uploaded 4 bytes to a pixel. When the upload operation is finished, the IEEE floats in ZRAM will be converted back to DSP floats. This double conversion can result in rounding errors. This option is only available when reading from DEV ZRAM BUFFER.

The following values are or'ed into the *mode* argument to specify which portion of Pixel Machine memory to upload from:

DEV FRONT BUFFER: Upload pixels from the front (currently displayed) portion of VRAM.

DEV\_BACK\_BUFFER: Upload pixels from the back (currently non-displayed) portion of VRAM.

DEV VRAMO BUFFER: Upload pixels from the VRAMO portion of VRAM.

DEV\_VRAM1\_BUFFER: Upload pixels from the VRAM1 portion of VRAM.

DEV ZRAM BUFFER: Upload pixels from ZRAM.

The sizes of the above buffers vary depending on the type of Pixel Machine being used as defined in the following table:

Model	FRONT	BACK	VRAM0	VRAM1	ZRAM
916	1024x1024	1024x1024	-		1024x1024
920	1280x1024	1280x1024	_	. —	1280x1024
932	1024x1024	1024x1024	1024x2048	1024x2048	1024x2048
940	1280x1024	1280x1024	1280x2048	1280x2048	1280x2048
964	2048x1024	2048x1024	2048x2048	2048x2048	2048x2048
964X	2048x1024	2048x1024	2048x2048	2048x2048	2048x2048

Note that when uploading from ZRAM, the number of "pixels" per scan line varies with the size of a pixel. For example, on a 964, a scan line of DEV\_MONO\_PIXELS is 8192 (4\*2048) pixels wide, a scan line of DEV\_MONO\_16 PIXELS is 4096 (2\*2048) pixels wide.

### RETURNS

**DEVget\_scan\_line** returns 0 upon success and -1 on failure. **DEVget\_scan\_line** also sets **DEVerrno** and **DEVerr\_msg** upon failure. If **DEVget\_scan\_line** fails, **DEVerrno** will be set to one of the following values:

DEV\_ERR\_INVPARAMETER: one or more of the parameters passed to DEVget\_scan\_line is invalid.

DEV\_ERR\_NORESPONSE: DEVget\_scan\_line sent a system command to the Pixel Machine to begin uploading but received no response from the pixel nodes. Typically this means that the pixel node programs did not call PMenable() to allow processing of the system command or the system command was not passed through the pipe nodes.

#### NOTES

**DEVget\_scan\_line** sends a system command to all pixel nodes to initiate uploading of the scan line. Pixel node programs must be prepared to receive this command or **DEVget\_scan\_line** will fail. The pixel node program should call **PMenable** with the **PM\_ENABLE GET SCAN LINE**,

2

PM\_ENABLE\_GET\_VRAM or PM\_ENABLE\_GET\_ZRAM argument during its initialization and should call PMgetcmd in its main processing loop. PMgetcmd will recognize the system command and call the appropriate routine to upload the scan line(s). In addition, the pipe node programs must make sure the system command is forwarded through each of the pipe nodes. The PMgetop function will transparently pass these system commands through to the pixel nodes.

**DEVget\_scan\_line** is an optimized version of **DEVget\_pixels** for operations like image upload and image processing.

**DEVget\_scan\_line** will be slightly faster if the scan line starts and ends on a subscreen boundary (i.e.,  $((x \% \text{ DEVx\_scale(system)}) == 0)$  and  $((x+npixl \% \text{ DEVx\_scale(system)}) == 0))$ .

SEE ALSO

DEVget\_pixels(3S) DEVinit(3H) PMenable(3N) PMgetcmd(3X)

DEVinit - opens and initializes Pixel Machine device

# SYNOPSIS

#include <host/devtools.h>

**DEVpixel\_system \*DEVinit()** 

# DESCRIPTION

**DEVinit** opens and initializes the device associated with the Pixel Machine. It should always be included at the start of any DEVtools host program.

**DEVinit** performs the following operations: opens the device, sets global variables that can be used to access the system configuration information, halts the processors, resets the processors and configures the pipes, and sets the pixel mode register.

**DEVinit** handles the signals **SIGHUP**, **SIGINT**, and **SIGTERM**. If any of these signals are received, the processors are stopped, the FIFOs are reset, and the pipe is restored to its original configuration.

**DEVinit** returns a pointer to the system descriptor if all of the operations complete successfully. If the operation fails, it returns NULL.

SEE ALSO

DEVopen(3S)

DEVload color tables - reads file of gamma calibration values and sets color lookup tables

## **SYNOPSIS**

#include <host/devtools.h>

void DEVload\_color\_tables(pixel\_system, filename)
DEVpixel\_system \*pixel\_system;
char \*filename;

## DESCRIPTION

**DEVload\_color\_tables** reads a file of gamma calibration values and sets the color lookup tables appropriately. The gamma file consists of a series of lines of the format:

*x.x y.y* 

Where:

x.x is a calibration level y.y is the measured video output

DEVload\_color\_tables computes color table values by interpolating the input values.

# NOTES

**DEVload\_color\_tables** is automatically called by **DEVinit** and **DEVopen** when the HYPER\_GAMMA environment variable is set.

SEE ALSO

DEVinit(3S) DEVopen(3S)

**DEVlock** - manage Pixel Machine locks

# SYNOPSIS

#include <host/devtools.h>

int DEVlock(key,device) int key; DEVpixel device \*device;

## DESCRIPTION

**DEVlock** is used to manage the locks for the Pixel Machine to prevent more than one user from accessing the machine at the same time. *key* designates the action desired; it must have one of the following values:

DEV\_KEYLOCK\_ASSIGN: assigns the device to a user DEV\_KEYLOCK\_UNASSIGN: clears a previous assignment DEV\_KEYLOCK\_LOCK: locks the device for a user DEV\_KEYLOCK\_UNLOCK: unlocks the device

Locking and assigning are similar processes, differing only in that locking has higher precedence. Locking is used by the **hyplock** and **hypfree** commands, while assigning is used by the **DEVopen** and **DEVclose** functions. The difference in precedence levels allows a user to lock a system using the **hyplock** command, run one or more programs that use **DEVopen** and **DEVclose** and still have the system locked upon completion of the programs. This may be useful to avoid having the contents of the screen corrupted, even after the program that created the image has completed.

**DEVopen**, **DEVopen** system – make a Pixel machine available to a user program

**SYNOPSIS** 

#include <host/devtools.h>

**DEVpixel system \*DEVopen()** 

DEVpixel\_system \*DEVopen\_system(options)
int options;

### DESCRIPTION

**DEVopen** makes a Pixel Machine available to a user program. The environment variables HYPER\_UNIT, HYPER\_ADDRESS, HYPER\_MODEL, HYPER\_PIPE HYPER\_GAMMA, and HYPER\_VIDEO are used to determine which machine is to be used and the configuration of the system.

If the device is already open, it is closed before **DEVopen** attempts to reopen it. **DEVopen** looks for a lock file for the device being requested. If the device is already locked, **DEVopen** returns NULL. Otherwise, a lock file is created to prevent the device from being accessed by another user.

If the open operation is successful, DEVopen returns a pointer to a system description block, otherwise NULL is returned.

The actual process of opening the device consists of:

creating a lock file for the desired device

opening the VME bus device associated with the Pixel Machine designated by the environment variable HYPER UNIT

allocating a memory area that is mapped to the device that has been opened

initializing a system description block that contains the memory map addresses for each of the boards and each of the processors in the Pixel Machine

configuring the pipes based on the contents of the HYPER PIPE environment variable

initializing the pixel mode registers on the pixel boards

configuring the video controller based on the contents of the HYPER\_MODEL, HYPER GAMMA and HYPER VIDEO environment variables.

The following system status information is updated by **DEVopen**:

- The color tables are updated based on the HYPER\_GAMMA environment variable. If HYPER\_GAMMA is set and is not null, it is used as the the name of a file that contains a gamma correction table. If HYPER\_GAMMA is not set or is null, a linear ramp is loaded into the color tables. If HYPER\_GAMMA does not contain an absolute pathname, it is used as a filename in the \$HYPER PATH/crts directory. Relative pathnames are not supported.

- The video control parameters are set based on the HYPER\_MODEL and HYPER\_VIDEO environment variables. The HYPER\_VIDEO variable contains a string that is parsed to produce a value that is passed to DEVset\_video\_options(). The string in HYPER\_VIDEO must be of the format:

sync source={int,ext}

sync\_on\_green={on,off}

The value after the equal sign must be one of the values listed in braces. The first value is the default; spaces in the string are ignored.

## **EXAMPLES**

HYPER VIDEO="sync source=ext sync on green=off"

HYPER VIDEO="sync source = int"

### NOTES

**DEVinit** is ordinarily used instead of **DEVopen**. **DEVopen** is provided for users who require lower level control of the Pixel Machine.

**DEVopen\_system** is identical to **DEVopen**, with the exception that an option parameter is provided to override certain default actions described above.

*options* must be zero or the value DEV\_OPEN\_NOCONFIG. Setting the *noconfig* option causes **DEVopen\_system** to suppress the steps that set the configuration of the machine. The steps omitted are:

- configuring the pipes
- initializing the pixel mode registers
- loading the color tables
- setting the video options

The *noconfig* option is used by commands like **devprint** and **hypstat** that need to access the Pixel Machine without altering the mode that the machine is running.

This function should only be used for applications that require lower level access to the machine.

#### SEE ALSO

**DEVload** color tables(3S)

**DEVpipe boot** – load a Pixel Machine executable into specified set of pipe nodes

## **SYNOPSIS**

#include <host/devtools.h>

options;

### DESCRIPTION

int

**DEVpipe\_boot** determines whether the specified Pixel Machine executable file has been loaded into the Pixel Machine. If the file has not been loaded, **DEVpipe\_boot** loads it into the specified set of nodes.

*pixel\_system* is a pointer to the system structure of the system to be loaded. *first\_node* and *last\_node* specify the range of nodes to be loaded. Setting *first\_node* to DEV\_ALL causes all of the pipe nodes to be loaded. *load\_table* is a pointer to an array of boolean values that indicate for each node whether or not the node should be loaded. If *first\_node* <= *node* <= *last\_node* and *load\_table[node]* is true, then the node is loaded. The load table feature is supplied to make it possible to load the same program into an arbitrary group of nodes while only reading the executable file once. If the load table feature is not needed, a null pointer can be used as the argument.

*options* is used to specify certain optional processing. This value must be zero or a bitwise or of one or more of the following values:

DEV BOOT VERBOSE: causes a description of the actions being performed to be displayed

DEV\_BOOT\_FORCE: causes the file to always be loaded regardless of the contents of the system status file

DEV\_BOOT\_CHECK\_TIME: causes the modification time of the file to be compared with the modification time of the file currently loaded into the node (if the filenames are the same). If the times are not the same, the file is reloaded.

#### RETURNS

**DEVpipe\_boot** returns zero if the operation was successful, -1 if an error occurred. The following error codes can be generated by **DEVpipe\_boot**:

DEV ERR LDFILEOPEN: the specified file could not be opened

DEV ERR LDFILERR: the specified file is not a valid object file

DEV ERR OTHER: miscellaneous error while loading the program

**DEVpipe** get – read a stream of bytes from the PIR of a pipe DSP

#### **SYNOPSIS**

#include <host/devtools.h>

int DEVpipe\_get(pixel\_system, node, buffer, nbytes, timeout)
DEVpixel\_system \*pixel\_system;
int node;
DEVbyte \*buffer;
int nbytes;
int timeout;

### DESCRIPTION

**DEVpipe\_get** reads a stream of bytes from the PIR of a pipe DSP. This function differs from **DEVpipe\_read** in that it requires a program running on the DSP to load data into the PIR register. The implementation differs in that:

it does not use DMA

the address from which the data is to be read cannot be supplied

a timeout parameter must be supplied

*pixel\_system* is a pointer to the system descriptor, *node* is the number of the pipe node from which the data is to be read. *buffer* points to the location into which the data is to be read. *nbytes* is the number of bytes of data to be read. *nbytes* should always be an even number. If *nbytes* is odd, *nbytes*+1 bytes of data will be read. *timeout* contains the number of times, for each two bytes transferred, that the PCR register is to be tested to see if the data has been sent successfully.

No byte order translation is performed. The data read will be in the same byte order as it is in the DSP memory.

As a result of this operation, the parallel communications modes are altered to set the interrupt vector to 16-bit mode.

NOTES

**DEVpipe\_get** returns the number of characters read.

The *timeout* parameter contains the number of loop iterations to be attempted before giving up. Because the execution rate depends on the system load, this could yield different results under different system load conditions. Also, because there is no sleep involved, the host process could consume a great deal of CPU time if the delay for each character is significant.

**DEVpipe get msg** – read a message from the PIR of a pipe DSP

## SYNOPSIS

#include <host/devtools.h>

int DEVpipe\_get\_msg(pixel\_system, node, buffer, nbytes, swap)
DEVpixel\_system \*pixel\_system;
int node;
DEVbyte \*buffer;
int nbytes;
int swap;

### DESCRIPTION

**DEVpipe\_get\_msg** reads a message from the PIR of a pipe DSP. This function is similar to **DEVpipe get** with the following exceptions:

a timeout parameter is not supplied

a byte swapping parameter is provided to allow mapping of DSP values into host values

Like **DEVpipe\_get**, **DEVpipe\_get\_msg** does not use DMA and requires that a program running on the DSP load the data into the PIR.

*pixel\_system* is a pointer to the system descriptor, *node* is the number of the pipe node from which the data is to be read. *buffer* points to the location into which the data is to be read. *nbytes* is the number of bytes of data to be read. *nbytes* should always be an even number. If *nbytes* is odd, *nbytes*+1 bytes of data will be read. *swap* must be one of the following values:

DEV SWAP NONE – no byte order conversion

DEV\_SWAP\_SHORT - the buffer is treated as a collection of 2-byte values and the bytes are ordered as required

DEV\_SWAP\_LONG - the buffer is treated as a collection of 4-byte values and the bytes are ordered as required.

If *swap* is DEV\_SWAP\_LONG, *nbytes* should be a multiple of 4, because a multiple of 4 bytes will always be read.

As a result of this operation, the parallel communications modes are altered to set the interrupt vector to 16-bit mode.

### NOTES

**DEVpipe\_get\_msg** returns the number of characters read.

This routine will hang and use a lot of CPU time if the process on the DSP does not load the expected data into the PIR.

DEVpipe get pir - read the PIR register of a pipe DSP

#### **SYNOPSIS**

#include <host/devtools.h>

DEVushort DEVpipe\_get\_pir(pixel\_system,node) DEVpixel\_system \*pixel\_system int node:

### DESCRIPTION

**DEVpipe\_get\_pir** reads the PIR register of a pipe DSP. This function is a special version of **DEVpipe get msg** that always fetches two bytes without adjusting the byte order.

Like **DEVpipe\_get\_msg**, **DEVpipe\_get\_pir** does not use DMA and it requires that a program running on the DSP load the PIR.

*pixel\_system* is a pointer to the system descriptor, *node* is the number of the pipe node from which the data is to be read.

As a result of this operation, the parallel communications modes are altered to set the interrupt vector to 16-bit mode.

DEVpipe get pir returns the contents of the DSP's PIR register as an unsigned short integer.

### NOTES

This routine will hang and use a lot of CPU time if the process on the DSP does not load the expected data into the PIR.

**DEVpixel halt** – halt a pixel node processor

# SYNOPSIS

#include <host/devtools.h>

int DEVpixel\_halt(pixel\_system,node)
DEVpixel\_system \*pixel\_system;
int node;

# DESCRIPTION

**DEVpixel\_halt** halts a pixel node processor. After the processor has halted, the parallel communications modes are altered to:

enable interrupts

enable DMA

set PAR to be autoincremented on DMA

set the interrupt vector to 16-bit mode

## RETURNS

DEVpixel\_halt returns DEV\_ERR\_OK if the operations succeeds, DEV\_ERR\_FAIL otherwise.

DEVpixel id check – check status of node's ID

# SYNOPSIS

#include <host/devtools.h>
#include <host/crt0.h>

int DEVpixel\_id\_check(system,node,id)
DEVpixel\_system \*system;
int node;
DEVcrt0 id \*id;

### DESCRIPTION

DEVpixel\_id\_check is used to check whether a node's ID has been corrupted.

system is a pointer to the system description information returned by **DEVopen**. node is the number of the node to which the ID is to be written, and is also used as a node identification number.

**DEVpixel\_id\_check** uses the parameter *id* to return the node ID information to the caller. *id* is a pointer to a node identification block.

## RETURNS

This function returns DEV\_ERR\_OK if the operation is successful, otherwise an error value is returned. The possible error values are:

DEVERR ID: Node ID information is invalid

DEVERR NODE: Node number is invalid

### SEE ALSO

DEVpixel\_read(3S) DEVpixel\_write(3S)

**DEVpixel\_id\_print** – read and print the node ID of a processor

# SYNOPSIS

#include <host/devtools.h>
#include <host/crt0.h>

int DEVpixel\_id\_print(system,node,id)
DEVpixel\_system \*system;
int node;
DEVcrt0 id \*id;

## DESCRIPTION

**DEVpixel\_id\_print** reads and prints the node ID of a processor's memory, and the node status information from the system status file and displays the information on standard output. **DEVpixel\_id\_print** reads the node ID from a processor and displays the information on standard output.

system is a pointer to the system descriptor. *node* is the number of the node to which the ID is to be written and is also used as a node identification number.

The checksum information in the node is compared with the value stored in the system status file on the host. If the checksum values do not match the message Node checksum does not match is printed beneath the program name.

This function returns DEV\_ERR\_OK if the operation is successful, otherwise an error value is returned. The possible error values are:

DEV\_ERR\_ID: Node ID information is invalid

DEV\_ERR\_NODE: Node number is invalid

# EXAMPLE

Pixel node 0 identification data:

. .

node	id:	0
crt0	format:	DEVtools
x nodes:		5
y noo	des:	4
x off	fset:	0
y off	Eset:	0
program:		/usr/xyz/prog.dsp
semaphore:		0

### SEE ALSO

DEVpixel\_write(3S) DEVpixel\_read(3S)

**DEVpipe\_put** – write a block of data to a pipe DSP's PDR register

### SYNOPSIS

#include <host/devtools.h>

int DEVpipe\_put(pixel\_system, node, buffer, nbytes, timeout)
DEVpixel\_system \*pixel\_system;
int node;
DEVbyte \*buffer;
int nbytes;
int timeout;

### DESCRIPTION

**DEVpipe\_put** writes a block of data to a pipe DSP's PDR register. This function differs from **DEVpipe\_write** in that it requires a program running on the DSP to read data from the PDR register and store it in the appropriate memory location. The implementation differs in that:

it does not use DMA

the address to which data is to be sent is not supplied

a timeout parameter must be supplied

*pixel\_system* is a pointer to the system descriptor, *node* is the number of the pipe node from which the data is to be written. *buffer* points to the data to be sent. *nbytes* is the number of bytes of data to be written. *nbytes* should always be an even number. If *nbytes* is odd, *nbytes*+1 bytes of data will be written. *timeout* contains the number of times, for each two bytes transferred, that the PCR register is to be tested to see if the data has been sent successfully.

No byte order translation is performed. The data sent will be in the same byte order as it is in buffer.

As a result of this operation, the parallel communications modes are altered to:

disable DMA

set PAR to not be autoincremented on DMA

set the interrupt vector to 16-bit mode.

DEVpipe put returns the number of characters written.

## NOTES

The *timeout* parameter contains the number of loop iterations to be attempted before giving up. Because the execution rate depends on the system load, this could yield different results under different system load conditions. Also, because there is no sleep involved, the host process could consume a great deal of CPU time if the delay for each character is significant.

**DEVpipe\_run** – begin execution of programs loaded into specified pipe nodes

# SYNOPSIS

#include <host/devtools.h>

void DEVpipe\_run(pixel\_system, first\_node, last\_node, options)
DEVpixel\_system \*pixel\_system;
int first\_node;
int last\_node;
int options;

# DESCRIPTION

DEVpipe\_run begins execution of the programs loaded into the specified pipe nodes.

*pixel\_system* is a pointer to the system structure of the system whose node is to be started. *first\_node* and *last\_node* specify the range of nodes.

options is used to specify certain optional processing. This value should be zero or the value DEV RUN VERBOSE, which causes DEVpipe run to provide additional information.

# RETURNS

**DEVpipe\_run** returns zero if execution was started successfully, -1 if an error occurred. The following error code can be generated by **DEVpipe\_run**:

DEV\_ERR\_STARTERR: the program loaded in the node could not be started

# NOTES

**DEVrun** can be used to begin execution on all pipe and pixel nodes.

# SEE ALSO

DEVrun(3H)

**DEVpipe\_write** – write a buffer to a pipe DSP

## SYNOPSIS

#include <host/devtools.h>

int DEVpipe\_write(pixel\_system, node, addr, buffer, nbytes)
DEVpixel\_system \*pixel\_system;
int node;
DEVushort addr;
DEVbyte \*buffer;
int nbytes;

## DESCRIPTION

DEVpipe\_write writes a buffer to a pipe DSP. The data is transferred using parallel DMA.

*pixel\_system* is a pointer to the system descriptor, *node* is the number of the pipe node from which the data is to be written. *addr* is the location in the DSP address space to which the data is to be sent. *buffer* points to the data to be sent. *nbytes* is the number of bytes of data to be written. *nbytes* should always be an even number. If *nbytes* is odd, *nbytes*+1 bytes of data will be written.

The data sent will be in the same byte order as it is in *buffer*. No byte order translation is performed.

**DEVpipe\_write** uses parallel DMA I/O to transfer the data. As a result, the parallel control register is updated by this routine. The parallel communications modes are altered to:

enable DMA

set PAR to be autoincremented on DMA

set the interrupt vector to 16 bit mode

## RETURNS

DEVpipe write should always return zero.

If *nbytes* is odd, **DEVpipe\_write** will write *nbytes*+1 bytes of data and return -1 as its return value. The return value should probably be the number of bytes written, not zero.

DEVpixel\_boot - load a Pixel Machine executable into specified set of pixel nodes

## **SYNOPSIS**

#include <host/devtools.h>

### DESCRIPTION

**DEVpixel\_boot** determines whether the specified Pixel Machine executable file has been loaded into the Pixel Machine. If the file has not been loaded, **DEVpixel\_boot** loads it into the specified set of nodes.

*pixel\_system* is a pointer to the system structure of the system to be loaded. *first\_node* and *last\_node* specify the range of nodes to be loaded. If *first\_node* is set to DEV\_ALL, all pixel nodes will be loaded. *load\_table* is a pointer to an array of boolean values that indicate for each node whether or not the node should be loaded. If *first\_node* <= *node* <= *last\_node* and *load\_table[node]* is true, then the node is loaded. The load table feature is supplied to make it possible to load the same program into an arbitrary group of nodes, while only reading the executable file once. If the load table feature is not needed, a null pointer can be used as the argument.

options is used to specify certain optional processing. This value must be zero or a bitwise or of one or more of the following values:

DEV\_BOOT\_VERBOSE: causes a description of the actions being performed to be displayed

DEV\_BOOT\_FORCE: causes the file to always be loaded regardless of the contents of the system status file

DEV\_BOOT\_CHECK\_TIME: causes the modification time of the file to be compared with the modification time of the file currently loaded into the node (if the filenames are the same). If the times are not the same, the file is reloaded.

### RETURNS

**DEVpixel\_boot** returns zero if the operation was successful, -1 if an error occurred. The following error codes can be generated by **DEVpixel\_boot**:

DEV\_ERR\_LDFILEOPEN: the specified file could not be opened DEV\_ERR\_LDFILERR: the specified file is not a valid object file DEV\_ERR\_OTHER: miscellaneous error while loading the program

DEVpixel\_buffer - selects the frame buffer to be displayed

# SYNOPSIS

#include <host/devtools.h>
#include <host/pixel.h>

void DEVpixel\_buffer(system,buffer)
DEVpixel\_system \*system;
DEVushort buffer;

# DESCRIPTION

DEVpixel buffer selects the frame buffer to be displayed.

system is a pointer to the system description information returned by **DEVopen**. buffer indicates which frame buffer is to be displayed, and must be one of the following values:

DEV_VBUF0:	Display frame buffer 0
DEV_VBUF1:	Display frame buffer 1

# NOTES

Because this function updates the pixel node flag registers, it should only be used when the Pixel Machine is halted.

**DEVpixel get** – read a stream of bytes from a pixel DSP's PIR register

### SYNOPSIS

#include <host/devtools.h>

int DEVpixel\_get(pixel\_system, node, buffer, nbytes, timeout)
DEVpixel\_system \*pixel\_system;
int node;
DEVbyte \*buffer;
int nbytes;
int timeout;

## DESCRIPTION

**DEVpixel\_get** reads a stream of bytes from a pixel DSP's PIR register. This function differs from **DEVpixel\_read** in that it requires a program running on the DSP to load data into the PIR register. The implementation differs in that:

it does not use DMA

the address from which the data is to be read cannot be supplied

a timeout parameter must be supplied

*pixel\_system* is a pointer to the system descriptor, *node* is the number of the pixel node from which the data is to be read. *buffer* points to the location into which the data is to be read. *nbytes* is the number of bytes of data to be read. *nbytes* should always be an even number. If *nbytes* is odd, *nbytes*+1 bytes of data will be read. *timeout* contains the number of times, for each two bytes transferred, that the PCR register is to be tested to see if the data has been sent successfully.

No byte order translation is performed. The data read will be in the same byte order as it is in the DSP memory.

As a result of this operation the parallel communications modes are altered to set the interrupt vector to 16-bit mode.

## RETURNS

DEVpixel\_get returns the number of characters read.

## NOTES

The *timeout* parameter contains the number of loop iterations to be attempted before giving up. Because the execution rate depends on the system load, this could yield different results under different system load conditions. Also, because there is no sleep involved, the host process could consume a great deal of CPU time if the delay for each character is significant.

**DEVpixel\_get\_msg** – read a message from a pixel DSP's PIR register

# SYNOPSIS

#include <host/devtools.h>

int DEVpixel\_get\_msg(pixel\_system, node, buffer, nbytes, swap)
DEVpixel\_system \*pixel\_system;
int node;
DEVbyte \*buffer;
int nbytes;
int swap;

#### DESCRIPTION

**DEVpixel\_get\_msg** reads a message from a pixel DSP's PIR register. This function is similar to **DEVpixel get** with the following exceptions:

a timeout parameter is not supplied

a byte swapping parameter is provided to allow mapping of DSP values into host values

Like **DEVpixel\_get**, **DEVpixel\_get\_msg** does not use DMA and requires that a program running on the DSP load the data into the PIR.

*pixel\_system* is a pointer to the system descriptor, *node* is the number of the pixel node from which the data is to be read. *buffer* points to the location into which the data is to be read. *nbytes* is the number of bytes of data to be read, and it should always be an even number. If *nbytes* is odd, *nbytes*+1 bytes of data will be read. *swap* must be one of the following values:

### **DEV SWAP NONE:**

no byte order conversion

# **DEV SWAP SHORT:**

the buffer is treated as a collection of 2-byte values and the bytes are ordered as required

### DEV SWAP LONG:

the buffer is treated as a collection of 4-byte values and the bytes are ordered as required.

If *swap* is DEV\_SWAP\_LONG, *nbytes* should be a multiple of 4 because a multiple of 4 bytes will always be read.

As a result of this operation, the parallel communications modes are altered to set the interrupt vector to 16-bit mode.

# RETURNS

DEVpixel\_get\_msg returns the number of characters read.

## NOTES

This routine will hang and use a lot of CPU time if the process on the DSP does not load the expected data into the PIR.

**DEVpixel get pir** - read the PIR register of a pixel DSP

### SYNOPSIS

#include <host/devtools.h>

DEVushort DEVpixel\_get\_pir(pixel\_system,node) DEVpixel\_system \*pixel\_system; int node;

# DESCRIPTION

**DEVpixel\_get\_pir** reads the PIR register of a pixel DSP. This function is a special version of **DEVpixel get msg**, and it always fetches two bytes without adjusting the byte order.

Like DEVpixel\_get\_msg, DEVpixel\_get\_pir does not use DMA and it requires that a program running on the DSP load the PIR.

*pixel\_system* is a pointer to the system descriptor, *node* is the number of the pixel node from which the data is to be read.

As a result of this operation, the parallel communications modes are altered to set the interrupt vector to 16-bit mode.

DEVpixel get pir returns the contents of the DSP's PIR register as an unsigned short integer.

### NOTES

This routine will hang and use a lot of CPU time if the process on the DSP does not load the expected data into the PIR.

**DEVpipe\_halt** – halt a pipe node processor

SYNOPSIS

#include <host/devtools.h>

int DEVpipe\_halt(pixel\_system,node)
DEVpixel\_system \*pixel\_system;
int node;

## DESCRIPTION

**DEVpipe\_halt** halts a pipe node processor. After the processor has halted, the parallel communications modes are altered to:

enable interrupts

enable DMA

set PAR to be autoincremented on DMA

set the interrupt vector to 16-bit mode

### NOTES

DEVpipe\_halt returns DEV\_ERR\_OK if the operations succeeds, DEV\_ERR\_FAIL otherwise.
**DEVpipe id check** – check status of node's ID

**SYNOPSIS** 

#include <host/devtools.h>
#include <host/crt0.h>

int DEVpipe\_id\_check(system,node,id)
DEVpixel\_system \*system;
int node;
DEVcrt0 id \*id;

## DESCRIPTION

**DEVpipe** id check is used to check whether a node's ID has been corrupted.

system is a pointer to the system description information returned by **DEVopen**. *node* is the number of the node to which the ID is to be written and is also used as a node identification number.

**DEVpipe\_id\_check** uses the parameter *id* to return the node ID information to the caller. *id* is a pointer to a node identification block.

This function returns DEV\_ERR\_OK if the operation is successful, otherwise an error value is returned. The possible error values are:

DEVERR ID: Node ID information is invalid

DEVERR NODE: Node number is invalid

SEE ALSO

DEVpipe\_write(3S) DEVpipe\_read(3S)

**DEVpipe\_id\_print** – read and print the node ID of a processor

#### **SYNOPSIS**

#include <host/devtools.h>
#include <host/crt0.h>

int DEVpipe\_id\_print(system,node,id)
DEVpixel\_system \*system;
int node;
DEVcrt0 id \*id;

### DESCRIPTION

**DEVpipe\_id\_print** reads the node ID from the processor's memory and the node status information from the system status file, and displays the information on standard output. **DEVpipe\_id\_print** reads the node ID from a processor and displays the information on standard output.

system is a pointer to the system descriptor. *node* is the number of the node to which the ID is to be written and is also used as a node identification number.

The checksum information in the node is compared with the value stored in the system status file on the host. If the checksum values do not match, the message Node checksum does not match is printed beneath the program name.

This function return DEV\_ERR\_OK if the operation is successful, otherwise an error value is returned. The possible error values are:

DEV ERR ID: Node ID information is invalid

DEV ERR NODE: Node number is invalid

# EXAMPLE

Pipe node 0 identification data:

```
node id: 0
crt0 format: DEVtools
x nodes: 5
y nodes: 4
x offset: 0
y offset: 0
program: /usr/xyz/prog.dsp
semaphore: 0
```

### SEE ALSO

DEVpipe\_write(3S) DEVpipe\_read(3S)

DEVpixel\_id\_write - write a node id block to a reserved location in a pixel node DSP's memory

### **SYNOPSIS**

#include <host/devtools.h>
#include <host/crt0.h>

int DEVpixel\_id\_write(system,node,name)
DEVpixel\_system \*system;
int node;
char \*name;

## DESCRIPTION

**DEVpixel\_id\_write** writes a node identification block to a reserved location in pixel node memory. The memory used to hold the node ID is allocated by the routine crt0, therefore pixel\_crt0.0 must be linked as part of the executable code running on the processor in order to use **DEVpixel id write**.

system is a pointer to the system description information returned by **DEVopen**. *node* is the number of the node to which the ID is to be written, and is also used as a node identification number. *name* is a pointer to the name that is to be assigned to the node.

## RETURNS

This function returns DEV\_ERR\_OK if the operation is successful, otherwise an error value is returned. The possible error values are:

DEVERR ID: Node ID information is invalid

DEVERR NODE: Node number is invalid

## SEE ALSO

DEVpixel\_write(3S)

DEVpixel\_mode\_init - initialize pixel board mode register

## SYNOPSIS

#include <host/devtools.h>

void DEVpixel\_mode\_init(system,omode)
DEVpixel\_system \*system;
DEVpixel modereg omode;

## DESCRIPTION

**DEVpixel\_mode\_init** sets overlay mode and initializes the gate bits, video shift rate, and the serial I/O connector selection fields in the pixel mode register on each pixel node board. A copy of the pixel mode register is maintained on the host because the board's pixel mode register cannot be read. As a result, **DEVpixel\_mode\_init** must be called during the initialization process, otherwise when a call is made that updates the pixel mode register (**DEVserial\_direction** for example), it will load the register with an uninitialized value.

system is a pointer to the system description information returned by **DEVopen**. omode must contain one of the following values:

DEV OVERLAY OFF: Uses the values in rgb

DEV\_OVERLAY\_ON: If any overlay bit is on, the overlay value is used for the red, green, and blue values. If all of the overlay bits are on, the inverse of rgb is used.

DEV\_OVERLAY\_FORCE: The overlay value is always used.

DEV\_OVERLAY\_MASK: If overlay bit 7 is on, the overlay value is used for red, green, and blue; otherwise rgb is used.

**DEVpixel\_mode\_init** initializes the other components of the pixel mode register to default values. The defaults are:

Mode Bits - DEV\_GATES\_SYNC | DEV\_GATES\_FIFO

Video shift rate - Appropriate value based on the system type

Serial I/O - Serial I/O connector/direction zero selected

### NOTES

In order for overlaying to be performed, the overlay flags must be set in both pixel node boards' pixel mode registers and in the individual processor's flag registers.

### SEE ALSO

DEVpixel\_overlay(3S) PMoverlay(3X)

**DEVpixel mode overlay -** set overlay mode in the pixel mode register

### SYNOPSIS

#include <host/devtools.h>

void DEVpixel\_mode\_overlay(system,omode)
DEVpixel\_system \*system;
DEVpixel modereg omode;

## DESCRIPTION

**DEVpixel\_mode\_overlay** sets the overlay mode in the pixel mode register of each pixel node board. Other fields of the pixel mode register are not affected.

system is a pointer to the system description information returned by **DEVopen**. omode must contain one of the following values:

DEV OVERLAY OFF: Uses the values in rgb

DEV OVERLAY ON: If any overlay bit is on, the overlay value

is used for the red, green, and blue values. If all of the overlay bits are on, the inverse of rgb is used.

## DEV OVERLAY FORCE: The overlay value is always used.

DEV\_OVERLAY\_MASK: If overlay bit 7 is on, the overlay value is used for red, green, and blue; otherwise rgb is used.

## NOTES

In order for overlaying to be performed, the overlay flags must be set in both the pixel node boards' pixel mode register and in the individual processor's flag registers.

### SEE ALSO

DEVpixel\_overlay(3S) PMoverlay(3X)

**DEVpixel\_overlay** – update overlay mode in all pixel processor's flag registers

# SYNOPSIS

#include <host/devtools.h>
#include <host/pixel.h>

void DEVpixel\_overlay(system,mode)
DEVpixel\_system \*system;
DEVushort mode;

## DESCRIPTION

**DEVpixel\_overlay** updates the overlay mode associated with each of the individual pixel processor's flag registers. The overlay mode must be set both in the pixel node board's pixel mode register and for the individual processors.

system is a pointer to the system description information returned by **DEVopen**. mode contains the new contents of the overlay flag and must be one of the following values:

DEV OVERLAY: Set the overlay flag

0 (zero): Clear the overlay flag

# NOTES

Because this function updates the pixel node flag registers, it should only be used when the Pixel Machine is halted.

The PMoverlay function should be used to set the overlay mode during execution.

SEE ALSO

PMoverlay(3X)

**DEVpixel\_put** – send a block of data to a pixel DSP's PDR register

### SYNOPSIS

#include <host/devtools.h>

int DEVpixel\_put(pixel\_system, node, buffer, nbytes, timeout)
DEVpixel\_system \*pixel\_system;
int node;
DEVbyte \*buffer;
int nbytes;
int timeout;

### DESCRIPTION

**DEVpixel\_put** sends a block of data to a pixel DSP's PDR register. This function differs from **DEVpixel\_write** in that it requires that a program running on the DSP read data from the PDR register and store it in the appropriate memory location. The implementation differs in that:

it does not use DMA

the address to which data is to be sent is not supplied

a timeout parameter must be supplied

*pixel\_system* is a pointer to the system descriptor, *node* is the number of the pixel node from which the data is to be written. *buffer* points to the data to be sent. *nbytes* is the number of bytes of data to be written. *nbytes* should always be an even number. If *nbytes* is odd, *nbytes*+1 bytes of data will be written. *timeout* contains the number of times, for each two bytes transferred, that the PCR register is to be tested to see if the data has been sent successfully.

No byte order translation is performed. The data sent will be in the same byte order as it is in buffer.

As a result of this operation, the parallel communication modes are altered to:

disable DMA

set PAR to not be autoincremented on DMA

set the interrupt vector to 16-bit mode

## RETURNS

**DEVpixel put** returns the number of characters written.

## NOTES

The *timeout* parameter contains the number of loop iterations to be attempted before giving up. Because the execution rate depends on the system load, this could yield different results under different system loadconditions. Also, because there is no sleep involved, the host process could consume a great deal of CPU time if the delay for each character is significant.

DEVpixel\_read - read a block of memory from a pixel DSP

## SYNOPSIS

#include <host/devtools.h>

int DEVpixel\_read(pixel\_system, node, addr, buffer, nbytes)
DEVpixel\_system \*pixel\_system;
int node;
DEVushort addr;
DEVbyte \*buffer;
int nbytes;

## DESCRIPTION

**DEVpixel\_read** reads a block of memory from a pixel DSP. The data is retrieved from DSP memory using parallel DMA.

*pixel\_system* points to the system descriptor, *node* is the number of the pixel node from which the data is to be read. *addr* is the location in the DSP address space that contains the data to be read. *addr* must be an even memory location, aligned on a 16-bit word boundary. *buffer* points to the location into which the data is to be read. *nbytes* is the number of bytes of data to be read. *nbytes* should always be an even number. If *nbytes* is odd, *nbytes*+1 bytes of data will be read.

No byte order translation is performed. The data read will be in the same byte order as it is in the DSP memory.

**DEVpixel\_read** uses parallel DMA I/O to transfer the data. As a result, the parallel control register is updated by this routine. The parallel communications modes are altered to:

enable DMA

set PAR to be autoincremented on DMA

set the interrupt vector to 16-bit mode

## RETURNS

DEVpixel\_read should always return zero.

If *nbytes* is odd, **DEVpixel\_read** reads *nbytes*+1 bytes of data and returns -1 as its return value. The return value should be the number of bytes written, not zero.

## SEE ALSO

DEVpixel\_get(3S)

**DEVpixel\_run** – begin execution of programs loaded into specified pixel nodes

## SYNOPSIS

#include <host/devtools.h>

void DEVpixel\_run(pixel\_system, first\_node, last\_node, options)
DEVpixel\_system \*pixel\_system;
int first\_node;
int last\_node;
int options;

## DESCRIPTION

DEVpixel\_run begins execution of the programs loaded into the specified pixel nodes.

*pixel\_system* is a pointer to the system structure of the system whose node is to be started. *first\_node* and *last node* specify the range of nodes.

options is used to specify certain optional processing. This value should be zero or the value DEV\_RUN\_VERBOSE, which will cause DEVpixel\_run to provide additional information.

# RETURNS

**DEVpixel\_run** returns zero if execution started successfully, -1 if an error occurred. The following error code can be generated by **DEVpixel\_run**:

DEV ERR STARTERR: the program loaded in the node could not be started

# NOTES

**DEVrun** can be used to begin execution on all pipe and pixel nodes.

DEVpixel\_system DEVpipe\_nodes, DEVlast\_pipe, DEVpixel\_nodes, DEVlast\_pixel, DEVx\_nodes, DEVy\_nodes, DEVx\_scale, DEVy\_scale, DEVx\_screen, DEVy\_screen, DEVmodel\_code, DEVvideo\_code, DEVpipe\_code - macros used to fetch system description information from the system descriptor

## SYNOPSIS

#include <host/devtools.h>

int DEVpipe\_nodes(pixel\_system)
DEVpixel\_system \*pixel\_system;

int DEVlast\_pipe(pixel\_system)
DEVpixel\_system \*pixel\_system;

int DEVpixel\_nodes(pixel\_system)
DEVpixel\_system \*pixel\_system;

int DEVlast\_pixel(pixel\_system)
DEVpixel system \*pixel system;

int DEVx\_nodes(pixel\_system)
DEVpixel\_system \*pixel system;

int DEVy\_nodes(pixel\_system)
DEVpixel\_system \*pixel\_system;

int DEVx\_scale(pixel\_system)
DEVpixel\_system \*pixel\_system;

int DEVy\_scale(pixel\_system) DEVpixel system \*pixel system;

int DEVx\_screen(pixel\_system)
DEVpixel\_system \*pixel\_system;

int DEVy\_screen(pixel\_system)
DEVpixel\_system \*pixel\_system;

DEVushort DEVmodel\_code(pixel\_system) DEVpixel system \*pixel system;

DEVushort DEVvideo\_code(pixel\_system) DEVpixel system \*pixel system;

DEVushort DEVpipe\_code(pixel\_system) DEVpixel\_system \*pixel system;

## DESCRIPTION

These macros are used to fetch system description information from the system descriptor. These macros should always be used to access this information. Direct use of the fields of the system structure is unsupported.

## **DEVtools**

The following describes the value returned by each macro:

DEVpipe nodes: the number of pipe node processors (0, 9, or 18).

DEVlast pipe: the number of the last pipe node. Useful for calling routines such as DEVpoll nodes.

DEVpixel nodes: the number of pixel node processors (16, 20, 32, 40, or 64).

DEVlast pixel: the number of the last pixel node. Useful for calling routines such as DEVpoll nodes.

DEVx nodes: the number of nodes in the X dimension (4, 5, 8, or 10)

DEVy\_nodes: the number of nodes in the Y dimension (4 or 8)

DEVx scale: the number of virtual nodes in the X dimension (8, or 10)

DEVy scale: the number of virtual nodes in the Y dimension (8)

DEVx\_screen: the screen width in pixels

DEVy\_screen: the screen hight in pixels

DEVmodel\_code: the system model

DEV\_MODEL\_916 DEV\_MODEL\_920 DEV\_MODEL\_932 DEV\_MODEL\_940 DEV\_MODEL\_964 DEV\_MODEL\_964X

DEVvideo\_code - the video mode in use DEV\_MODEL\_VIDEO\_HIRES DEV\_MODEL\_VIDEO\_NTSC DEV\_MODEL\_VIDEO\_PAL

DEVpipe\_code – the pipe mode in use DEV\_MODEL\_PIPE\_SINGLE DEV\_MODEL\_PIPE\_PARALLEL DEV\_MODEL\_PIPE\_SERIAL DEV\_MODEL\_PIPE\_NONE

**DEVpixel write** – write a buffer to a pixel DSP

### **SYNOPSIS**

#include <host/devtools.h>

int DEVpixel\_write(pixel\_system, node, addr, buffer, nbytes)
DEVpixel\_system \*pixel\_system;
int node;
DEVushort addr;
DEVbyte \*buffer;
int nbytes;

## DESCRIPTION

DEVpixel write writes a buffer to a pixel DSP. The data is transferred using parallel DMA.

*pixel\_system* is a pointer to the system descriptor, *node* is the number of the pixel node from which the data is to be written. *addr* is the location in the DSP address space to which the data is to be sent. *addr* must be an even memory location, aligned on a 16-bit word boundary. *buffer* points to the data to be sent. *nbytes* is the number of bytes of data to be written. *nbytes* should always be an even number. If *nbytes* is odd, *nbytes*+1 bytes of data will be written.

No byte order translation is performed. The data sent will be in the same byte order as it is in buffer.

**DEVpixel\_write** uses parallel DMA I/O to transfer the data. As a result, the parallel control register is updated by this routine. The parallel communications modes are altered to:

enable DMA

set PAR to be autoincremented on DMA

set the interrupt vector to 16-bit mode

# RETURNS

DEVpixel\_write should always return zero.

If *nbytes* is odd, **DEVpixel\_write** writes *nbytes*+1 bytes of data and returns -1 as its return value. The return value should be the number of bytes written, not zero.

DEVpoll nodes - poll DSP processors for messages

### **SYNOPSIS**

#include <host/devtools.h>

int DEVpoll\_nodes(pixel\_system, firstpipe, lastpipe, firstpixel, lastpixel, iter\_cnt, sleep)
DEVpixel\_system \*pixel\_system;
int firstpipe, lastpipe;
int firstpixel, lastpixel;
int iter\_cnt;
int sleep;

#### DESCRIPTION

**DEVpoll\_nodes** is used to poll one, several, or all of the DSP processors to see if they have a *user* or *system message* to be served. **DEVpoll\_nodes** *must* be called if the user has calls to **printf**, **PMusermsg**, **PMsiodir** or **PMhost\_exit**, in a pipe or pixel node program.

*firstpipe* and *lastpipe* are the node numbers of the lowest and highest pipe node processors to be polled. *firstpixel* and *lastpixel* are the node numbers of the lowest and highest pixel node processors to be polled. The lowest node on a system is always zero; the highest number is the number of nodes minus one. If either the pipe or pixel nodes are not to be polled, DEV\_NONE should be supplied for both the first and last values.

*iter\_cnt* is the number of times the designated processors are to be polled. If *iter\_cnt* is DEV\_FOREVER, the polling process continues until an *exit* message is sent from one of the polled processors or until the host program is interrupted. An exit message can be sent from a processor by calling the **PMhost exit** function.

sleep is the amount of time to sleep between each time the processors are polled. All of the processors are polled before the system sleeps. If *sleep* is DEV\_NONE, no sleep call is made. The *sleep* value is passed to the **usleep** system call. DEV\_NONE should only be used for applications that require very fast response to Pixel Machine message requests because it causes the host to consume a large amount of CPU time.

#### RETURNS

**DEVpoll\_nodes** returns after all of the specified processors have been polled *iter\_cnt* times, or when an exit message is received from any of the polled nodes. The return value is 1 if an exit message was received, 0 if the specified number of iterations have been completed.

#### EXAMPLE

```
#include <host/devtools.h>
```

SEE ALSO

DEVexit(3H) DEVinit(3H) PMhost\_exit(3N) printf(3N) PMusermsg(3N) PMsiodir(3X) usleep(2) on the host system

DEVput\_color\_map - update color tables from video controller board and return the value

## **SYNOPSIS**

#include <host/devtools.h>

<pre>void DEVput_color_map(pixel_system, r, g, b)</pre>						
DEVpixel_system		<pre>*pixel_system;</pre>				
int	r[DEV_	VIDEO	TABLE];			
int	g[DEV]	VIDEO	TABLE];			
int	b[DEV]	VIDEO	TABLE];			

# DESCRIPTION

**DEVput\_color\_map** updates the color tables from the video controller board and returns the values to the caller. Each color table contains 256 entries; each entry is a 10-bit value (0-1023).

## SEE ALSO

DEVget\_color map(3S)

DEVput\_image\_header - write a Pixel Machine image header to a file

SYNOPSIS

#include <stdio.h>
#include <host/devtools.h>
#include <host/devimage.h>
#include <host/deverror.h>

int DEVput\_image\_header(file, image\_header, optional\_header) FILE \*file; DEVimage\_header \*image\_header; DEVbyte \*optional\_header;

## DESCRIPTION

**DEVput\_image\_header** writes the DEVimage\_header and the optional user header (if one exists) to the specified file.

*file* is a file descriptor obtained from a previous call to *fopen(3)*. The file must have been successfully opened for writing and the file pointer should be pointing to the beginning of the file (i.e., no previous writes have been issued). Upon return from **DEVput\_image\_header**, the file pointer will be set to where the pixel data should start (i.e., past the image and optional headers).

**DEVput\_image\_header** will convert the DEVimage\_header structure pointed to by *image\_header* into a string of decimal ASCII characters and write it to the file pointed to by *file*. If the *magic* structure member is 0, it will be set to DEV\_IMAGE\_MAGIC before being written. If *magic* is non-zero, it will be written as is.

If *optional\_header* is non-zero, the characters pointed to it will be written to *file* immediately after the image header. *image header->optional header size* bytes will be written.

### RETURNS

**DEVput\_image\_header** returns 0 upon success and -1 on failure. **DEVput\_image\_header** will set **DEVerrno** to indicate the reason for failure:

DEV\_ERR\_BAD\_MAGIC: The magic number is not DEV IMAGE MAGIC.

DEV\_ERR\_WRITE\_ERR: An error was returned by the *fwrite(3)* system call while writing either the image header or the optional header.

## NOTES

No value in the **DEVimage header** should be greater than 100,000,000.

### SEE ALSO

DEVimage\_header(4) DEVget image header(3S)

**DEVput pixel, DEVput pixels** – write pixels into the frame buffer

### **SYNOPSIS**

#include <host/devtools.h>

void DEVput\_pixel(system,buffer,x,y,r,g,b,o)
DEVpixel\_system \*system;
int buffer, x, y;
short r, g, b, o;

void DEVput\_pixels(system,buffer,x,y,r,g,b,o,npixl)
DEVpixel\_system \*system;
int buffer, x, y;
short \*r, \*g, \*b, \*o;
int npixl;

# DESCRIPTION

**DEVput\_pixel** writes pixels into a frame buffer. Through this routine, a program can update the Pixel Machine frame buffer without having to deal with the details of how the frame memory is organized on different models of the system.

system is a pointer to the system description information returned by **DEVinit**. buffer indicates which frame buffer is to be updated (must be the value 0 or 1). x is the x coordinate, y is the y coordinate. r, g, b, and o are the values to be stored in the red, green, blue, and overlay values in the frame buffer.

**DEVput\_pixels** writes a sequence of pixels for a single scan line. npixl is the number of pixels to be written. r, g, b, and o point to a sequence of red, green, blue, and overlay values to be written.

#### NOTES

DEVput\_scan\_line provides a more efficient and flexible facility for dowloading image data.

### SEE ALSO

DEVpixel\_write(3S) DEVput\_scan\_line(3H)

DEVput scan line - download an image or a portion of an image to a Pixel Machine

#### **SYNOPSIS**

#include <host/devtools.h>
#include <host/devimage.h>

int DEVput\_scan\_line(system, x, y, npixels, nlines, mode, pixel\_buffer)
DEVpixel\_system \*system;
unsigned int x, y;
unsigned int npixels, nlines, mode;
DEVpixel \*pixel buffer;

#### DESCRIPTION

**DEVput\_scan\_line** transfers an image or a portion of an image from the host to a Pixel Machine. The data is transferred from the host memory area specified by *pixel\_buffer* according to the mode specified by *mode*.

system is a pointer to the system description information returned by **DEVinit**. x is the starting x screen coordinate, y is the starting y screen coordinate.

The buffer pointed to by *pixel\_buffer* must contain (*npixels* \* *nlines* \* pixel size) bytes, where the pixel size is determined by the *mode* argument as described below. In all cases, pixels will be accessed in *pixel\_buffer* in the following order: (x,y), (x+1,y), ..., (x+npixels-1,y), (x,y+1), ..., (x+npixels-1,y+nlines-1).

The *mode* argument is used to specify two pieces of information: how the pixels are stored in the array pointed to by *pixel\_buffer*, and which portion of Pixel Machine memory the data should be copied to. These values are or'ed into the *mode* argument. Valid pixel format values for *mode* are:

DEV\_RGBA\_PACKED\_PIXELS: pixels are stored in *pixel\_buffer*, 4 bytes to a pixel, in the following order: red, green, blue, alpha.

DEV\_RGB\_PACKED\_PIXELS: pixels are stored in *pixel\_buffer*, 3 bytes to a pixel, in the following order: red, green, blue.

DEV\_MONO\_R\_PIXELS: pixels are stored in *pixel\_buffer*, 1 byte to a pixel, with the red component of the pixel actually being stored. This option is not available when downloading to DEV ZRAM BUFFER.

DEV\_MONO\_G\_PIXELS: pixels are stored in *pixel\_buffer*, 1 byte to a pixel, with the green component of the pixel actually being stored. This option is not available when downloading to DEV ZRAM BUFFER.

DEV\_MONO\_B\_PIXELS: pixels are stored in *pixel\_buffer*, 1 byte to a pixel, with the blue component of the pixel actually being stored. This option is not available when downloading to DEV\_ZRAM BUFFER.

DEV\_MONO\_A\_PIXELS: pixels are stored in *pixel\_buffer*, 1 byte to a pixel, with the alpha (overlay) component of the pixel actually being stored. This option is not available when downloading to DEV ZRAM BUFFER.

DEV\_MONO\_PIXELS: pixels are stored in *pixel\_buffer*, 1 byte to a pixel. When downloading to VRAM, the 1 byte pixel is written to the red, green, and blue component of a pixel.

DEV\_MONO\_16\_PIXELS: pixels are stored in *pixel\_buffer*, 2 bytes to a pixel. This option is only available when downloading to DEV ZRAM BUFFER.

DEV DSP FLOAT PIXELS: pixels are stored in *pixel buffer*, 4 bytes to a pixel. This option

is only available when downloading to DEV ZRAM BUFFER.

DEV\_IEEE\_FLOAT\_PIXELS: pixels are stored in *pixel\_buffer*, 4 bytes to a pixel. During the download operation, the pixels are converted from IEEE format floating point to DSP floating point. This option is only available when downloading to DEV ZRAM BUFFER.

The following values are or'ed into the *mode* argument to specify which portion of Pixel Machine memory to download to:

DEV\_FRONT\_BUFFER: Download pixels to the front (currently displayed) portion of VRAM.

DEV\_BACK\_BUFFER: Download pixels to the back (currently non-displayed) portion of VRAM.

DEV VRAM0 BUFFER: Download pixels to the VRAM0 portion of VRAM.

DEV VRAM1 BUFFER: Download pixels to the VRAM1 portion of VRAM.

DEV ZRAM BUFFER: Download pixels to ZRAM.

The sizes of the above buffers vary depending on the type of Pixel Machine being used as defined in the following table:

Model	FRONT	BACK	VRAM0	VRAM1	ZRAM
916	1024x1024	1024x1024	-	-	1024x1024
920	1280x1024	1280x1024	-	-	1280x1024
932	1024x1024	1024x1024	1024x2048	1024x2048	1024x2048
940	1280x1024	1280x1024	1280x2048	1280x2048	1280x2048
964	2048x1024	2048x1024	2048x2048	2048x2048	2048x2048
964X	2048x1024	2048x1024	2048x2048	2048x2048	2048x2048

Note that subscreens are not used when downloading to ZRAM.

RETURNS

**DEVput\_scan\_line** returns 0 upon success and -1 on failure. **DEVput\_scan\_line** also sets **DEVerrno** and **DEVerr msg** upon failure.

NOTES

**DEVput\_scan\_line** sends a series of system commands to the pipe and pixel nodes to perform the download operation. Pixel node programs must be prepared to receive this command or **DEVput\_scan\_line** will fail. The pipe node programs must use **PMgetop** to read command opcodes. The download commands are implicitly copied through the pipe by **PMgetop**. The pixel node program should call **PMenable** during its initialization and should call **PMgetcmd** in its main processing loop. **PMgetcmd** will recognize the system command and call the appropriate routine to display the scan line(s).

**DEVput\_scan\_line** is an optimized version of **DEVput\_pixels** for operations like image upload and image processing.

SEE ALSO

DEVput\_pixels(3S) DEVinit(3H) PMenable(3N) PMgetcmd(3X) PMgetop(3P)

**DEVread** z - read a buffer of bytes from the Z memory of a pixel node

## SYNOPSIS

#include <host/devtools.h>

void DEVread\_z(pixel\_system, node, x, y, buffer, n)
DEVpixel\_system \*pixel\_system;
int node;
int x, y;
DEVbyte \*buffer;
int n;

## DESCRIPTION

**DEVread\_z** reads a buffer of bytes from the Z memory of a pixel node. *pixel\_system* is a pointer to the memory mapped control block of the processor whose memory is to be read. x and y are the coordinates in the Z memory where the read operation starts. *buffer* is a pointer to the area into which the data is to be read. n is the number of bytes to be read.

The Z memory is organized as 256 rows of 256 32-bit words. "x" is the row from which the data is to be read, "y" is the word offset of the data to be read. An even number of bytes is always read.

Transfers must not attempt to wrap past the end of a row, or, in other words, the offset in bytes (y \* 4) plus the number of bytes read (n) must not exceed the number of bytes per row (1024).

### NOTES

This routine does not perform any byte order changes.

## SEE ALSO

DEVpixel\_read(3S) DEVget\_scan\_line(3H)

**DEVrelease\_pipe\_semaphore**, **DEVrelease\_pixel\_semaphore** - clear the software semaphore in the memory of one of the DSP processors

## **SYNOPSIS**

#include <host/devtools.h>

void DEVrelease\_pipe\_semaphore(pixel\_system,node)
DEVpixel\_system \*pixel\_system;
int node;

void DEVrelease\_pixel\_semaphore(pixel\_system,node)
DEVpixel\_system \*pixel\_system;
int node;

## DESCRIPTION

These routines are used to clear the software semaphore in the memory of one of the DSP processors. *pixel\_system* is a pointer to the system descriptor, *node* is the number of the pipe or pixel node whose semaphore is to be reset.

The semaphore can be set by a program running on one of the nodes by calling the PMsetsem routine.

These routines are used by the message serving system, but may also be used by user applications that do not make use of the message serving routines. They should never be called by routines that serve message requests from the Pixel Machine, as this would effect the synchronization between the Pixel Machine and host system.

DEVrun - begin execution of all pipe and pixel nodes

## SYNOPSIS

void DEVrun(pixel\_system) DEVpixel system \*pixel system;

# DESCRIPTION

**DEVrun** is used to begin execution of the programs loaded into all pipe and pixel node processors. *pixel\_system* is the system pointer returned by **DEVinit**. **DEVinit** must be called before calling **DEV-run**. If **DEVpipe boot** and **DEVpixel boot** are used, they must be called before calling **DEVrun**.

## SEE ALSO

DEVinit(3H) DEVpipe\_boot(3H) DEVpixel\_boot(3H)

DEVserial\_direction - updates the serial I/O link direction

SYNOPSIS

#include <host/devtools.h>

int DEVserial\_direction(system, direction)
DEVpixel\_system \*system;
int direction;

# DESCRIPTION

DEVserial direction updates the serial I/O link direction.

system is a pointer to the system description information returned by **DEVopen**. direction indicates the direction in which data is to be transferred, and must be one of:

DEV\_NORTH DEV\_EAST DEV\_SOUTH DEV\_WEST

Based on the system type, the appropriate calls to **DEVpixel\_mode\_serial** are executed to configure the system for the desired serial I/O direction.

# RETURNS

Returns 0 on success.

SEE ALSO

devprint(1) DEVpoll\_nodes(3M) PMsiodir(3X)

DEVshadow\_off - turns off updating of color lookup tables from shadow tables

## SYNOPSIS

#include <host/devtools.h>

void DEVshadow\_off(pixel\_system)
DEVpixel\_system \*pixel\_system;

# DESCRIPTION

**DEVshadow\_off** turns off updating of the color lookup tables from the shadow tables. To avoid flickering caused by partially updated color tables, this function should be called before updating the lookup tables.

## SEE ALSO

DEVshadow\_on(3S)

**DEVswap\_long** – convert from DSP32 long integers to host long integers

# SYNOPSIS

#include <host /devtools.h>

# void

DEVswap\_long(buffer, nbyte) DEVbyte \*buffer; int nbyte;

## DESCRIPTION

**DEVswap\_long** converts an array of long integers in DSP32 format to long integers in the host format(and vice-versa). The pointer to the array is passed in the argument *buffer*. The size of the array in bytes is passed in the argument *nbyte*. *nbyte* is not the number of elements in the array.

The conversion is done in place.

SEE ALSO

DEVswap\_short(3S) DEVdsp\_ieee(3S) DEVieee\_dsp(3S) DEVsswapl(3S)

**DEVswap\_pipe** – switch primary and alternate pipes of a dual pipe system

# SYNOPSIS

#include <host/devtools.h>

void DEVswap\_pipe()

### DESCRIPTION

On a dual pipe system, with the pipes operating in parallel mode, one pipe is the primary pipe and the other is the alternate pipe. **DEVswap\_pipe** reverses the functions of the two pipes. This is used to balance the load between the two pipes.

**DEVswap\_pipe** sends a *system* command to the primary pipe to perform the broadcast bus arbitration. The command is passed through each of the pipe nodes until it reaches the last pipe node. When the last pipe node processes the swap-pipe command, it releases the broadcast bus to the alternate pipe. It then requests the bus and waits for bus access to be granted.

#### NOTES

Programs in pipe nodes 8 and 17 must have called PMenable(PM\_ENABLE\_SWAP\_PIPE) in order to correctly respond to the *system* command that DEVswap\_pipe() sends.

Pipe node programs must use **PMgetop** to read command opcodes. The swap-pipe commands are implicitly copied through the pipe by **PMgetop**.

Pipe node programs can control the broadcast bus independently using the PMswap pipe function.

SEE ALSO

PMenable(3N) PMgetop(3P) PMbus\_wait(3P) PMswap pipe(3P)

-

# NAME

DEVswap\_short - convert from DSP32 short integers to host short integers

## SYNOPSIS

#include <host/devtools.h>

# void

DEVswap\_short(buffer, nbyte) DEVbyte \*buffer; int nbyte;

## DESCRIPTION

**DEVswap\_short** converts an array of short integers in DSP32 format to short integers in the host format (and vice-versa). The pointer to the array is passed in the argument *buffer*. The size of the array in bytes is passed in the argument *nbyte*. *nbyte* is not the number of elements in the array.

The conversion is done in place.

## SEE ALSO

DEVswap\_long(3S) DEVdsp\_ieee(3S) DEVieee\_dsp(3S)

DEVuser\_msg\_enable - define a message code and specify functions to be called

### **SYNOPSIS**

#include <host/devtools.h>
#include <host/msgserve.h>

int DEVuser msg enable(code, pipefunction, pixelfunction)

int code;

int (\*pipefunction)(),

(\*pixelfunction)();

## DESCRIPTION

**DEVuser\_msg\_enable** allows a program to define a message code that is to be recognized by the polling routine, and to specify the functions that are to be called to service the message.

*code* is the user message code. It must be greater than zero, but must be less than the value DEV HIGHEST USER MESSAGE (defined in host/msgserve.h).

When a user message with the value *code* is received from a DSP, the polling routine will call **pipefunction** if the message is from a pipe node, or **pixelfunction** if the message is from a pixel node.

pipefunction must be defined as:

int pipefunction(opcode, pixel\_system, node)
int opcode;
DEVpixel\_system \*pixel\_system;
int node;

pixelfunction must be defined as:

```
int pixelfunction(opcode, pixel_system, node)
int opcode;
DEVpixel_system, *pixel_system;
int node;
```

opcode is the value of code; this allows one function to service several codes. pixel\_system is the system descriptor. node is the node number of the processor that sent the message.

#### SEE ALSO

DEVpoll\_nodes(3H) PMusermsg(3N)

DEVwait\_exit - wait for pixel nodes to signal completion, then call DEVexit

# SYNOPSIS

void DEVwait\_exit()

# DESCRIPTION

**DEVwait\_exit** sends a *system* command to all pixel nodes informing them that the host wishes to exit. The pixel node programs must have called **PMenable** with the PM\_ENABLE\_WAIT\_EXIT argument at initialization in order to process the *system* command correctly.

Upon receipt of the *system* command, the pixel nodes perform a **PMpsync** operation to ensure all nodes have finished, then sends a message to the host. When the host sees this message, it automatically calls **DEVexit** before returning to the user.

## SEE ALSO

DEVclose(3S) DEVinit(3H) DEVexit(3H) PMenable(3N)

**DEVwrite**, **DEVcwrite**, **DEVwrite**, **DEVcwrite**, **DEVwrite\_alt**, **DEVcread**, **DEVreadn**, **DEVreadn\_alt**, - macros to write to the Pixel Machines pipelines and read commands back from the feedback FIFO

## SYNOPSIS

#include <host/devtools.h>
#include <host/devcommand.h>

DEVulong DEVcommand(opcode,length) short opcode; short length;

short DEVcommand\_opcode(command)
long command;

short DEVcommand\_length(command)
long command;

void DEVcwrite0(command)
long command;

void DEVcwrite0\_alt(command)
long command;

void DEVcwrite1(command,type,x)
long command;
/\* type is the type name of the remaining arguments \*/
type x;

void DEVcwrite1\_alt(command,type,x)
long command;
/\* type is the type name of the remaining arguments \*/
type x;

void DEVwrite1(type,x)
/\* type is the type name of the remaining arguments \*/
type x;

void DEVwrite1\_alt(type,x)
/\* type is the type name of the remaining arguments \*/
type x;

void DEVcwrite2(command,type,x,y)
long command;
/\* type is the type name of the remaining arguments \*/
type x, y;

void DEVcwrite2\_alt(command,type,x,y)
long command;
/\* type is the type name of the remaining arguments \*/
type x, y;

void DEVwrite2(type,x,y) /\* type is the type name of the remaining arguments \*/ type х, у; void DEVwrite2 alt(type,x,y) /\* type is the type name of the remaining arguments \*/ type x, y; void DEVcwrite3(command,type,x,y,z) long command; /\* type is the type name of the remaining arguments \*/ type x, y, z; void DEVcwrite3 alt(command,type,x,y,z) long command; /\* type is the type name of the remaining arguments \*/ type x, y, z; void DEVwrite3(type,x,y,z) /\* type is the type name of the remaining arguments \*/ type x, y, z; void DEVwrite3 alt(type,x,y,z) /\* type is the type name of the remaining arguments \*/ type x, y, z; void DEVcwrite4(command,type,x,y,z,w) long command: /\* type is the type name of the remaining arguments \*/ type x, y, z, w; void DEVcwrite4 alt(command,type,x,y,z,w) long command; /\* type is the type name of the remaining arguments \*/ type x, y, z, w; void DEVwrite4(type,x,y,z,w) /\* type is the type name of the remaining arguments \*/ type x, y, z, w; void DEVwrite4\_alt(type,x,y,z,w) /\* type is the type name of the remaining arguments \*/ type x, y, z, w; void DEVcwrite5(command,type,a,b,c,d,e) long command; /\* type is the type name of the remaining arguments \*/ type a, b, c, d, e; void DEVcwrite5 alt(command,type,a,b,c,d,e) long command; /\* type is the type name of the remaining arguments \*/ a, b, c, d, e; type

```
void DEVwrite5(type,a,b,c,d,e)
/* type is the type name of the remaining arguments */
type
        a, b, c, d, e;
void DEVwrite5 alt(type,a,b,c,d,e)
/* type is the type name of the remaining arguments */
type
        a, b, c, d, e;
void DEVcwrite6(command,type,a,b,c,d,e,f)
        command:
long
/* type is the type name of the remaining arguments */
type
        a, b, c, d, e, f;
void DEVcwrite6 alt(command,type,a,b,c,d,e,f)
        command;
long
/* type is the type name of the remaining arguments */
type
        a, b, c, d, e, f;
void DEVwrite6(type,a,b,c,d,e,f)
/* type is the type name of the remaining arguments */
type
        a, b, c, d, e, f;
void DEVwrite6 alt(type,a,b,c,d,e,f)
/* type is the type name of the remaining arguments */
type
        a, b, c, d, e, f;
void DEVcwrite7(command,type,a,b,c,d,e,f,g)
long
        command;
/* type is the type name of the remaining arguments */
type
        a, b, c, d, e, f, g;
void DEVcwrite7 alt(command,type,a,b,c,d,e,f,g)
long
        command:
/* type is the type name of the remaining arguments */
type
        a, b, c, d, e, f, g;
void DEVcwrite8(command,type,a,b,c,d,e,f,g,h)
long
        command:
/* type is the type name of the remaining arguments */
type
        a, b, c, d, e, f, g, h;
void DEVcwrite8 alt(command,type,a,b,c,d,e,f,g,h)
long
        command:
/* type is the type name of the remaining arguments */
        a, b, c, d, e, f, g, h;
type
void DEVcwrite9(command,type,a,b,c,d,e,f,g,h,i)
long
        command;
/* type is the type name of the remaining arguments */
type
        a, b, c, d, e, f, g, h, i;
void DEVcwrite9 alt(command,type,a,b,c,d,e,f,g,h,i)
long
        command;
```

/\* type is the type name of the remaining arguments \*/
type a, b, c, d, e, f, g, h, i;

```
void DEVwrite9(type,a,b,c,d,e,f,g,h,i)
long command;
/* type is the type name of the remaining arguments */
type a, b, c, d, e, f, g, h, i;
```

```
void DEVwrite9_alt(type,a,b,c,d,e,f,g,h,i)
long command;
/* type is the type name of the remaining arguments */
type a, b, c, d, e, f, g, h, i;
```

```
void DEVcwriten(command,type,block,length)
long command;
/* type is the type name of block */
type block[];
int length;
```

void DEVcwriten\_alt(command,type,block,length)
long command;
/\* type is the type name of block \*/
type block[];
int length;

```
void DEVwriten(type,block,length)
/* type is the type name of block */
type block[];
int length;
```

void DEVwrite\_alt(type,block,length)
/\* type is the type name of block \*/
type block[];
int length;

void DEVcread0(command)
long command;

void DEVcread0\_alt(command)
long command;

void DEVreadn(type,block,length)
/\* type is the type name of block \*/
type block[];
int length;

void DEVreadn\_alt(type,block,length)
/\* type is the type name of block \*/
type block[];
int length;

#### DESCRIPTION

These macros are used to write commands to the Pixel Machine pipelines and to read commands back from the feedback FIFO.

Each command consists of a command code, an operand count, and a list of 32-bit operands. The operands can be integers, host floating point numbers, or Pixel Machine floating point numbers. The interpretation of the contents of the operands is the responsibility of the user written code on the Pixel Machine that interprets the commands.

Macros that end with the string \_alt write to the alternate pipe of a multi-pipe system, the routines without \_alt write to the primary pipe. \_alt macros must not be used on single pipe systems or on multi-pipe systems whose pipes are configured in parallel.

**DEVcommand** is used to encode an opcode and parameter count into a 32-bit command code. The *command* argument of the **DEVcwrite** macros is usually a call to **DEVcommand**.

DEVcommand\_opcode and DEVcommand\_length are used with the DEVreadn macros to extract the opcode and length from the encoded value.

The **DEVcwrite0** through **DEVcwrite9** macros are used to write commands and a number of operands that match the last character of the macro name. **DEVwrite0** through **DEVwrite9** macros write only operands, they do not output a command code.

The read and write macros contain a *type* argument. This indicates the type of the arguments being read or written. The storage class of the *type* argument must be such that sizeof(type) == 4 bytes and *type* is word aligned. All of the argument types in a given macro invocation must be the same. To create a command with four arguments, the first two of which are floats and the last two of which are ints, the following sequence of commands must be used:

DEVcwrite2(DEVcommand(opcode, 4), float, x, y); DEVwrite2(int, i, j);

The **DEV\_writen** and **DEV\_readn** macros are used to write and read a block of operands. *block* is an array of values to be used as operands. *length* is the number of elements of *block* to be used. *length* must be less then or equal to 64.

# NOTES

In a pipeless Pixel Machine, the **DEVwrite** macros write directly to the broadcast bus FIFOs. The **DEVread** and **alt** macros should not be used in a pipeless Pixel Machine.

**DEVwrite\_z** - writes a buffer of bytes into the Z memory of a pixel node

### **SYNOPSIS**

#include <host/devtools.h>

void DEVwrite\_z(pixel\_system,node, x, y, buffer, n)
DEVpixel\_system \*pixel\_system;
int node;
int x, y;
DEVbyte \*buffer;
int n;

## DESCRIPTION

**DEVwrite\_z** writes a buffer of bytes into the Z memory of a pixel node.  $pixel_system$  is a pointer to the system description information returned by **DEVopen()**. x and y are the coordinates in the Z memory where the write operation starts. *buffer* is a pointer to the data to be written. n is the number of bytes to be written.

The Z memory is organized as 256 rows of 256 32-bit words. "x" is the row to which the data is to be written, "y" is the word offset of the data to be written. An even number of bytes is always written.

Transfers must not attempt to wrap past the end of a row, or, in other words, the offset in bytes (y \* 4) plus the number of bytes written (n) must not exceed the number of bytes per row (1024).

# NOTES

This routine does not perform any byte order changes.

### SEE ALSO

DEVput\_scan\_line(3H) DEVopen(3S) DEVpixel write(3S)

**PMapply** – apply a function to all subscreens

## SYNOPSIS

#include <pxm.h>

void PMapply(function [,arg] ...)
void (\*function)(PMsubscrn \*scrn ...);

## DESCRIPTION

**PMapply** provides a convenient method of calling a rendering *function* once for each subscreen, independent of the Pixel Machine model the code is being run on. *function* must take a pointer to a subscreen structure as its first argument, which is inserted by **PMapply**; the other *args* given to **PMapply** are passed on unchanged in each call to *function* made by **PMapply**.

### **EXAMPLES**

To set a pixel node's image memory to a specified color using the DEVtools routine PMclear:

```
PMpixeltype color;
PMapply(PMclear, 0, 0, PMimax, PMjmax, &color);
```

Without **PMapply** the above call would have to be written:

```
PMclear(PMscrns[0],0,0,PMimax,PMjmax,&color);
if (PMmx)
{
     PMclear(PMscrns[1],0,0,PMimax,PMjmax,&color);
     if (PMmy)
     {
        PMclear(PMscrns[2],0,0,PMimax,PMjmax,&color);
        PMclear(PMscrns[3],0,0,PMimax,PMjmax,&color);
        }
}
```

Of course, if the user is not concerned with portability across different models of the Pixel Machine, neither **PMapply** nor the *if* statements are needed. In this case, specify 1, 2 or 4 calls to the required function (in this example **PMclear**) with the corresponding subscreen argument, depending on the number of subscreens in the model.

### NOTES

**PMapply** is only useful in calling routines that do not modify their arguments and whose return value is, not needed.
PMclear – fill a rectangular region of the screen

SYNOPSIS

#include <pxm.h>

void PMclear(scrn, imin, jmin, imax, jmax, color) PMsubscrn \*scrn; short imin, jmin; short imax, jmax; PMpixeltype \*color;

## DESCRIPTION

**PMclear** fills a retangular section of a pixel node's subscreen memory with *color*. *scrn* is a pointer to an initialized PMsubscrn structure.

imin, jmin, imax and jmax are subscreen coordinates with the legal ranges:

- i [0, PMimax]
- j [0, PMjmax]

**PMimax** and **PMjmax** are automatically initialized to the appropriate value for the current model (see the *DEVtools User's Guide* for more information on subscreen ranges).

Values beyond these ranges will generate unpredictable results.

*color* is a pointer to a PMpixeltype structure containing the red, green, blue and overlay components to **PMclear** the region to. Each pixel within the region bounded by *imin*, *imax*, *jmin*, and *jmax* will be set to these values.

## NOTES

Refer to PMzbrk(3X) for page register use.

PMcolor\_float - macro that converts internal color value to floating point number

## SYNOPSIS

#include <pxm.h>

float PMcolor\_float(color)
int color;

### DESCRIPTION

**PMcolor\_float** is a macro that converts an internal color value to a floating point number in the range 0.0 - 1.0.

.

#### SEE ALSO

PMint\_color(3N) PMcolor\_int(3N) PMfloat\_color(3N)

PMcolor\_int - macro that converts internal color value to an integer

#### **SYNOPSIS**

#include <pxm.h>

int PMcolor\_int(color)
int color;

## DESCRIPTION

PMcolor\_int is a macro that converts an internal color value to an integer in the range 0 - 255.

# SEE ALSO

PMcolor\_float(3N) PMfloat\_color(3N) PMint\_color(3N)

**PMcopy f** – fast but dangerous 32 bit D/VRAM copy

SYNOPSIS

void PMcopy\_f(to, from, count)
register float \*to, \*from;
register int count;

### DESCRIPTION

PMcopy\_f copies *count* words (4 bytes each) using a sequence of the longword-copy instruction:

a0 = (r3 + + = r4 + +) \* a0;

to reduce loop overhead.

to and from are any kind of pointer as long as they are 4 byte aligned. They can be pointers that use page registers. They will work properly as long as the appropriate page registers were correctly initialized.

For copying VRAM, it is necessary to call PMcopy\_f twice, once with RG pointers and once with BO pointers.

This is the most efficient copy available. **PMcopy\_f** calls *mover* which can copy up to 64 words with no overhead. *mover* resides in BANK 1 to eliminate conflict wait states in most cases.

For VRAM or DRAM to VRAM or DRAM copy, each 32-bit copy takes 550ns including clock stretching. For VRAM or DRAM to SRAM (and vice-versa) each copy is 375ns. For SRAM it takes 200ns plus any possible conflict wait states. If both pointers point to BANK 0 (.text section or automatic data), there are no wait states. If one pointer is in BANK 1, there is one 50ns conflict wait state, two if both pointers point to BANK 1. All global and static data generated by the C compiler reside in BANK 1 by default. Loop overhead is only encountered every 64 words.

#### NOTES

This copy is so blindingly fast that it may interfere with the video shift register load temporarily messing up the display. This problem only occurs in VRAM; it is perfectly safe in SRAM.

### RETURNS

Results are undefined if the to and from pointers overlap.

If count < 1 it will be treated as a 1.

### SEE ALSO

PMcopy\_s(3X) PMcopy\_v(3X)

**PMcopy s** – safe 32–bit DRAM or VRAM copy

**SYNOPSIS** 

void PMcopy\_s(to, from, count)
register float \*to, \*from;
register int count;

#### DESCRIPTION

**PMcopy s** copies *count* words (4 bytes each) using a 2 instruction loop.

to and from are any kind of pointer, but they must be 4 byte aligned. They can be pointers that use page registers. They will work properly as long as the appropriate page registers were correctly initialized.

For copying VRAM, it is necessary to call PMcopy\_s twice, once with RG pointers and once with BO pointers.

This copy is a little slower than **PMcopy\_f**, but is guaranteed not to cause any video flashing problems. It also resides in BANK 1 to eliminate conflict wait states in most cases.

For VRAM or DRAM to VRAM or DRAM copy, each 32-bit copy takes 550ns including clock stretching. For VRAM or DRAM to SRAM (and vice-versa) each copy is 375ns. For SRAM it takes 200ns plus any possible conflict wait states. If both pointers point to video bank 0 (.text section or automatic data), there are no wait states. If one pointer is in video bank 1, there is one 50ns conflict wait state, two if both pointers point to video bank 1. All global and static data generated by the C compiler reside in video bank 1 by default. Add to these times 200ns for loop overhead per word.

#### NOTES

Results are undefined if the to and from pointers overlap or are not 4 byte aligned.

If *count* < 1 **PMcopy** s will return immediately.

### SEE ALSO

PMcopy\_f(3X) PMcopy\_v(3X)

**PMcopy**  $\mathbf{v}$  – 32-bit copy with variable increments

**SYNOPSIS** 

void PMcopy\_v(to, from, to\_inc, from\_inc, count)
register float \*to, \*from;
register int to\_inc, from\_inc;
register int count;

# DESCRIPTION

**PMcopy\_v** is similar to **PMcopy\_s** but it allows the user to specify the increments for both the *to* and *from* pointers. The 32-bit copy and both increments are all accomplished in one DSP32 instruction, plus one more instruction for loop control.

to and from can be any kind of pointer, but must be 4 byte aligned. to inc and from inc are the increments to be added to the pointers after each 32-bit copy.

count is the number of 4 byte words to copy.

### NOTES

Results are undefined if the to and from pointers overlap, or are not 4 byte aligned.

#### RETURNS

If count < 1 **PMcopy** v returns immediately.

SEE ALSO

PMcopy\_s(3X)

PMcopycmd - copy opcode, parameter count, and data from input to output FIFO of a pipe node

# SYNOPSIS

#include <pxm.h>

## void PMcopycmd()

#### DESCRIPTION

**PMcopycmd** copies the opcode, count, and parameters of a pipe command to the output FIFO. The parameters are copied directly from the input FIFO, but the opcode and count are copied from the **PMcommand** structure (which is initialized by a previous call to **PMgetop**).

# NOTES

PMcopycmd can only be called from a pipe node program.

## SEE ALSO

PMcommand(4N) PMgetop(3P) PMgetcmd(3X) PMgetdata(3P) PMputop(3P) PMputdata(3P)

1

PMcopyftob - copy front to back

**SYNOPSIS** 

#include <pxm.h>

void PMcopyftob(scrn, i, j, npix, nline)PMsubscrn\*scrn;inti, j;intnpix, nline;

#### DESCRIPTION

PMcopyftob copies a block of video memory from the front buffer to the back buffer.

scrn is a pointer to an initialized PMsubscrn structure. i and j are the starting location of the block to be copied. *npix* is the number of pixels and *nline* is the number of scan lines to be copied.

*i* and *j* are in the range of [0-PMimax] and [0-PMjmax], respectively. *npix* and *nlines* are in the range of [1-PMimax+1] and [1-PMjmax+1], respectively.

This function also works in single buffer mode.

### NOTES

Values outside these ranges will generate unpredictable results.

To copy from the back to front buffer call PMswapback before and after the call to PMcopyftob.

PMcopyftob saves and restores page registers.

SEE ALSO

PMswapback(3X) PMcopyvtov(3X)

PMcopyvtov - copy blocks of VRAM

### SYNOPSIS

#include <pxm.h>

void PMcopyvtov(bank\_from, bank\_to, i, j, ni, nj, di, dj)
int bank\_from, bank\_to;
int i, j;
int ni, nj;
int di,dj;

## DESCRIPTION

PMcopyvtov copies a block of video memory from the specified video bank bank\_from to bank\_to.

The banks can be PM\_VRAM0\_BUFFER or PM\_VRAM1\_BUFFER. You can copy from either bank to itself, or to the other bank.

i and j are the starting location of the block to be moved.

ni and nj are the number of pixels in the i and j directions, respectively, to be copied.

di and dj are the destination coordinates of the block.

i, j, di and dj are all in the range [0-255]; ni and nj are in the range [1-256].

## NOTES

A value of less than 1 is treated as 1. Values outside these ranges will generate unpredictable results.

#### SEE ALSO

PMcopy s(3N)

PMcopyvtoz - copy video RAM to DRAM

#### **SYNOPSIS**

#include <pxm.h>

void PMcopyvtoz(scrn, start\_i, start\_j, len\_i, len\_j, dest\_i, dest\_j, mode)
PMsubscrn \*scrn;
int start\_i;
int start\_j;
int len\_i;
int len\_j;
int dest\_i;
int dest\_j;
int mode;

#### DESCRIPTION

**PMcopyvtoz** copies a rectangular section of the VRAM buffer, *len\_i* pixels by *len\_j* pixels, from the processor space coordinates, *start\_i* and *start\_j*, to the Z (DRAM) buffer. *dest\_i* and *dest\_j* are the destination coordinates in DRAM, and correspond to *start\_i* and *start\_j*, respectively. The section that is copied depends on the value of *mode*. If the value is the defined constant PM\_FRONT\_BUFFER, the image will be copied from the front, or visible, buffer. If the value is the defined constant PM\_BACK\_BUFFER, the image will be copied from the scopied from the back, or invisible, buffer. Images in VRAM 1 can be copied by or'ing in the value PM\_VRAM1 BUFFER.

The pixel data is organized in ZRAM so that the color data is placed into four adjacent bytes, in the order red, green, blue, overlay. Pixels with the same value of y are stored in the same row of memory; those with the same value of x and same color, are stored in the same column. Each pixel that is owned by a processor is adjacent to the next pixel owned by that processor, regardless of subscreen. For example, ZRAM for a 964 will contain data for every eighth pixel in both x and y, while ZRAM for a 916 will contain the data for every fourth pixel in both x and y. Each pixel is copied to a different, well defined, location. Pixels do not overwrite each other.

### NOTES

It does not make sense to use PM\_VRAM1\_BUFFER on models 916 or 920 because screen pixels are already stored in both sections of VRAM.

This function can be called using **PMapply()**, which will call the function for all subscreens for each of the processor coordinates chosen. However, if the data to be copied does not align so that the upper left hand pixel falls on node 0 and subscreen 0, and the lower right hand pixel falls on the highest processor and highest subscreen, the function will need to be called more selectively. In that case the processor coordinates for each subscreen and the processors involved will need to be calculated from screen space, and this function called within each pixel node for each subscreen structure, with the appropriate arguments.

### SEE ALSO

PMcopyztov(3X) PMcopyztoz(3X) PMqcopyztoz(3X)

PMcopyztov - copy DRAM to video RAM

# SYNOPSIS

#include <pxm.h>

void PMcopyztov(scrn, start\_i, start\_j, len\_i, len\_j, dest\_i, dest\_j, mode)
PMsubscrn \*scrn;
int start\_i;
int start\_j;
int len\_i;
int len\_j;
int dest\_i;
int dest\_j;
int mode;

#### DESCRIPTION

**PMcopyztov** copies a rectangular section of DRAM, *len\_i* pixels wide by *len\_j* pixels high, starting at coordinates *start\_i* and *start\_j*, to the VRAM buffer. *dest\_i* and *dest\_j* are the destination coordinates in VRAM, and correspond to *start\_i* and *start\_j*, respectively. The section of VRAM which is copied to depends on the value of *mode*. If the value is the defined constant PM\_FRONT\_BUFFER, the image will be copied to the front, or visible, buffer. If the value is the defined constant PM\_BACK\_BUFFER, the image will be copied to the back, or invisible, buffer. These values may be or'ed with PM\_VRAM1 BUFFER to copy to VRAM1.

This function is the inverse of **PMcopyvtoz**(), and assumes that the data in DRAM has the structure that **PMcopyvtoz** would impose.

#### NOTES

It does not make sense to use PM\_VRAM1\_BUFFER on models 916 or 920 because screen pixels are already stored in both sections of VRAM.

### SEE ALSO

PMcopyvtoz(3X) PMcopyztoz(3X) PMqcopyztoz(3X)

PMcopyztoz - copy from one section of DRAM to another

### SYNOPSIS

#include <pxm.h>

void PMcopyztoz(start\_i, start\_j, len\_i, len\_j, dest\_i, dest\_j)
int start\_i;
int start\_j;
int len\_i;
int len\_j;
int dest\_i;
int dest\_j;

#### DESCRIPTION

**PMcopyztoz** copies a rectangular section of DRAM, with dimensions *len\_i* long words (4 byte units) by *len\_j* long words, starting at coordinates *start\_i* from *start\_j* to another section of DRAM buffer. *dest\_i* and *dest\_j* are the destination coordinates, and correspond to *start\_i* and *start\_j*, respectively.

The  $\_i$  arguments are in units of 4 bytes, e.g., 1 byte for each of red, green, blue and overlay, or the space for one float. Thus, if *start\_i* is set to 1, and *len\_i* is set to 2, 8 bytes will be copied on each row, starting at an offset of 4 bytes from the beginning of the row. In the  $\_j$  direction, one row is copied to one row.

### NOTES

This function provides a copy from one address to another, arbitrary, address. If there is no chance of overlapping copies, the function **PMqcopyztoz()** should be used, because it is faster and uses less code space.

#### SEE ALSO

PMqcopyztoz(3X) PMcopyvtoz(3X) PMcopyztov(3X)

PMcos - trigonometric function to compute the cosine of an angle

# SYNOPSIS

#include <libmath.h>

float PMcos(theta) float theta;

# DESCRIPTION

PMcos returns the cosine of *theta*.

theta must be in radians and be between -Pi/2 and +Pi/2.

**PMdblbuff** – enable double buffering mode

SYNOPSIS

#include <pxm.h>

### void PMdblbuff()

### DESCRIPTION

**PMdblbuff** enables double buffering. Double buffering implies a distinction between a visible buffer that is displayed by the video controller and a pixel buffer in which pixels are modified. **PMswapbuff** exchanges these two buffers.

PMsnglbuff disables double buffering.

## SEE ALSO

PMswapbuff(3X) PMsnglbuff(3X)

PMdelay - do nothing for a specified time

# SYNOPSIS

#include <pxm.h>

void PMdelay(time)
int time;

# DESCRIPTION

PMdelay executes a delay loop for (time / 250) seconds.

PMenable – enable processing of selected system commands

#### **SYNOPSIS**

#include <pxm.h>
#include <syscmd.h>

### void PMenable(function)

### DESCRIPTION

**PMenable** enables reception of certain *system* commands that are sent by host programs. After calling **PMenable**, any *system* commands that are generated by the host will be correctly processed when the pixel node receives them using **PMgetcmd**.

**PMenable** should be called as part of the program's initialization and *must* be called with one of the following #defines:

PM\_ENABLE\_GET\_SCAN\_LINE: Enables processing of all system commands sent by the **DEVget\_scan\_line** host routine. This option allows upload of pixels from both VRAM and ZRAM. This option only applies to pixel nodes.

PM\_ENABLE\_GET\_VRAM: Enables processing of *system* commands sent by the **DEVget\_scan\_line** host routine to upload pixels from VRAM only. This option saves space if ZRAM pixel upload is not needed. This option only applies to pixel nodes.

PM\_ENABLE\_GET\_ZRAM: Enables processing of *system* commands sent by the **DEVget\_scan\_line** host routine to upload pixels from ZRAM only. This option saves space if VRAM pixel upload is not needed. This option only applies to pixel nodes.

PM\_ENABLE\_PUT\_SCAN\_LINE: Enables processing of all system commands sent by the **DEVput\_scan\_line** host routine. This option allows download of pixels to both VRAM and ZRAM. This option only applies to pixel nodes.

PM\_ENABLE\_PUT\_VRAM: Enables processing of *system* commands sent by the **DEVput\_scan\_line** host routine to download pixels to any portion of VRAM. This option saves space if ZRAM pixel upload is not needed. This option only applies to pixel nodes.

PM\_ENABLE\_PUT\_ZRAM: Enables processing of *system* commands sent by the **DEVput\_scan\_line** host routine to download pixels to any portion of ZRAM. This option saves space if VRAM pixel upload is not needed. This option only applies to pixel nodes.

PM\_ENABLE\_SWAP\_PIPE: Enables processing of *system* commands sent by the **DEVswap\_pipe** host routine. This option only applies to pipe nodes, and should only be used by the last node of each parallel pipe (nodes 8 and 17).

PM\_ENABLE\_WAIT\_EXIT: This option allows processing of *system* commands sent by the **DEVwait\_exit** host function. This option applies only to pixel nodes.

If **PMenable** is *not* called before the host sends the *system* command, the *system* command will not be processed correctly.

### NOTES

PMenable is implemented as a macro.

It is important to enable only those functions that will actually be used, because each one takes up additional code space.

SEE ALSO

DEVget\_scan\_line(3H) DEVput\_scan\_line(3H) DEVswap\_pipe(3S) DEVwait\_exit(3H) PMgetcmd(3X)

**PMfb\_on** - direct output commands to the feedback FIFO **PMfb\_off** - direct output commands to the regular output FIFO

### SYNOPSIS

#include <pxm.h>

void PMfb\_on()
void Pbfb\_off()

# DESCRIPTION

PMfb\_on directs the output of subsequent PMputop, PMputdata, and PMcpycmd calls to the feedback FIFO instead of the output FIFO.

PMfb\_off redirects the output to the output FIFO instead of the feedback FIFO.

### NOTES

These functions must only be called from the last pipe node of each pipe board (nodes 8 and 17).

PMfdiv - perform floating point division

## SYNOPSIS

#include <libmath.h>
float PMfdiv(a, b)
float a, b;

## DESCRIPTION

**PMfdiv** computes the floating point value a \* (1.0 / b). If b is equal to zero, **PMfdiv** returns a large value of the same sign as a.

# NOTES

PMfdiv is intended to be called by assembly language routines.

PMfreezaddr - decrement references to a page register

SYNOPSIS

#include <pxm.h>

void PMfreezaddr(ptr)
char \*ptr;

# DESCRIPTION

**PMfreezaddr** is called to decrement the number of references to a page register. *ptr* is the pointer returned by a previous call to **PMgetzaddr**. The pointer may have been incremented and still work with **PMfreezaddr** as long as it did not get incremented past the end of the block.

Neither the contents of the page register nor the contents of the memory are changed in any way. The purpose of **PMfreezaddr** is to make the page register available for use when it is no longer needed to access this particular address, so that it may be used by a call to **PMgetzaddr** with a different PMzdesc descriptor.

#### NOTES

If **PMfreezaddr**() is called with the PMzdesc returned by **PMgetzaddr**(), and **PMgetzaddr**() is called again with the same PMzdesc, the value of the returned pointer may change, but the contents of the memory pointed to will not be changed.

#### SEE ALSO

PMgetzaddr(3X) PMgetzdesc(3X) PMzbrk(3X) PMblock\_reg(3X) PMavail\_reg(3X) PMset\_lowreg(3X) PMset\_hireg(3X)

PMfxtoi - map a linear function of x from screen space to processor space i

### **SYNOPSIS**

#include <pxm.h>

PMfxtoi(scrn, a, b) PMsubscrn \*scrn; float a, b;

### DESCRIPTION

**PMfxtoi** converts an expression of the form  $f(x)=A_{xy} x+B_{xy}$  to an expression of the form  $f(i)=A_{ij} i+B_{ij}$ . The macro actually modifies the values of A and B.

In the above expressions, the subscripts xy and ij are used to denote a constant in (x,y) space and a constant in (ij) space, respectively.

## NOTES

PMfxtoi is implemented as a macro.

### SEE ALSO

DEVtools User's Guide PMfxytoij(3X) PMfytoj(3X)

PMfxytoij - map a linear function of x and y from screen space to processor space i and j

### SYNOPSIS

#include <pxm.h>

PMfxytoij(scrn, a, b, c) PMsubscrn \*scrn; float a, b, c;

# DESCRIPTION

**PMfxyoij** converts an expression of the form  $f(x,y)=A_{xy} x+B_{xy} y+C_{xy}$  to an expression of the form  $f(i,j)=A_{ij} i+B_{ij} j+C_{ij}$ . The macro actually modifies the values of A, B and C.

In the above expressions, the subscripts xy and ij are used to denote a constant in (x,y) space and a constant in (ij) space, respectively.

## NOTES

PMfxytoij is implemented as a macro.

## SEE ALSO

DEVtools User's Guide PMfxtoi(3X) PMfytoj(3X)

PMfytoj - map a linear function of y from screen space to processor space j

### **SYNOPSIS**

#include <pxm.h>

PMfytoj(scrn, a, b) PMsubscrn \*scrn; float a, b;

# DESCRIPTION

**PMfytoj** converts an expression of the form  $f(y)=A_{xy} y+B_{xy}$  to an expression of the form  $f(j)=A_{ij} j+B_{ij}$ . The macro actually modifies the values of A and B.

In the above expressions, the subscripts xy and ij are used to denote a constant in (x,y) space and a constant in (ij) space, respectively.

### NOTES

PMfytoj is implemented as a macro.

## SEE ALSO

DEVtools User's Guide PMfxtoi(3X) PMfxytoij(3X)

**PMgetcmd** – load command from a pixel node FIFO

## SYNOPSIS

#include <pxm.h>

#### short PMgetcmd()

### DESCRIPTION

**PMgetcmd** reads an opcode, parameter count, and parameters from the input FIFO and stores them in the global **PMcommand** structure. The parameters are placed in the array pointed to by **PMcommand.data** ptr. The opcode is returned.

If the received command contains a negative opcode, the command is treated as a *system* command and the appropriate *system* function is invoked. If the appropriate *system* command has not been previously initialized by a call to **PMenable**, the command is ignored. In any case, **PMgetcmd** will consume all *system* commands until a user (positive opcode) command is read from the input FIFO.

## NOTES

**PMgetcmd** can only be called from a pixel node.

Unlike pipe nodes, pixel nodes may only receive commands from the FIFO.

**PMgetcmd** is implemented as a macro.

#### SEE ALSO

PMcommand(4N) PMenable(3N)

1

PMgetdata - get data from a pipe node FIFO

# SYNOPSIS

#include <pxm.h>

# void PMgetdata()

# DESCRIPTION

PMgetdata reads parameters of a command from the input FIFO. The parameters are placed in the array pointed to by PMcommand.data\_ptr.

## NOTES

PMgetdata can only be called from a pipe node.

PMgetdata must be preceded by a call to PMgetop.

SEE ALSO

PMcommand(4N) PMgetop(3P) PMputdata(3P)

PMgetop – get opcode and parameter count from input FIFO of a pipe node

## SYNOPSIS

#include <pxm.h>

short PMgetop()

### DESCRIPTION

**PMgetop** loads an opcode and parameter count from the input FIFO and stores them in the global **PMcommand** structure. It returns the opcode.

If the received command contains a negative opcode, the command is treated as a *system* command and the appropriate *system* function is invoked. If the appropriate *system* command has not been previously initialized by a call to **PMenable**, the command is passed on to the output FIFO of this pipe node. In any case, **PMgetop** will consume all *system* commands until a user (positive opcode) command is read from the input FIFO.

## NOTES

PMgetop can only be called from a pipe node.

PMgetop must be followed by a call to PMgetdata if PMcommand.count is non-zero.

**PMgetop** is implemented as a macro.

#### SEE ALSO

PMcommand(4N) PMenable(3N) PMgetdata(3P) PMputop(3P)

**PMgetpix** – read a pixel from the current buffer

SYNOPSIS

#include <pxm.h>

short \*PMgetpix(scrn, i, j, color)
PMsubscrn \*scrn;
short i, j;
PMpixeltype \*color;

### DESCRIPTION

**PMgetpix** reads a single pixel from the frame buffer. *scrn* is a pointer to an initialized **PMsubscrn** structure corresponding to the subscreen from which the pixel is read.

i and j are subscreen coordinates with the following legal ranges:

i [0, PMimax]

j [0, PMjmax]

**PMimax** and **PMjmax** are set to the appropriate value for the current model by system initialization. (see the *DEVtools User's Guide* for more information on subscreen ranges).

Values beyond these ranges will generate unpredictable results.

color is a pointer to a PMpixeltype structure whose red, green, blue and overlay components will be loaded with the pixel data contained at (i, j) in scrn.

**PMgetpix** returns a pointer to the next pixel on the given row (i+1,j). This pointer can be used by **PMqget** for more efficient frame buffer access.

### NOTES

Refer to PMzbrk(3X) for page register use.

### SEE ALSO

PMputpix(3X) PMqget(3X)

**PMgetrow, PMgetcol, PMputrow, PMputcol** – read or write a scanline or scancolumn from pixel memory without subscreens

#### **SYNOPSIS**

#include <pxm.h>

void PMgetrow(buf, row, col, npix) PMpixeltype \*buf; int row, col, npix;

void PMgetcol(buf, row, col, npix)
PMpixeltype \*buf;
int row, col, npix;

void PMputrow(buf, row, col, npix) PMpixeltype \*buf; int row, col, npix;

void PMputcol(buf, row, col, npix)
PMpixeltype \*buf;
int row, col, npix;

## DESCRIPTION

These four functions implement reading and writing pixels in *subscreen-independent* space. That is, these routines treat pixel memory as a single block of pixels and alternate access to subscreens as needed to preserve this illusion. Thus, in a 916, for example, instead of using **PMgetscan** and calling it four times with each of the four 128 by 128 subscreens, **PMgetrow** can be called once on a 256 by 256 buffer of pixels without the use of subscreens. This abstraction is useful for working in deinterleaved pixel space (e.g., filtering code). Either rows or columns can be accessed with these four functions.

For a full screen image, the size of subscreen-independent pixel memory is:

model	cols	rows
964X	160	128
964	128	128
940/932	128	256
920/916	256	256

PMgetrow and PMputrow read or write a row of pixels at a time, while PMgetcol and PMputcol read or write columns.

*buf* is a buffer of pixels to write to pixel memory (**PMputrow**, **PMputcol**) or read from pixel memory (**PMgetrow**, **PMgetcol**). The *buf* array must be large enough to store the requested pixels.

col and row are coordinates in subscreen-independent space. The number of pixels is specified in npix. Note that each pixel will take up 8 bytes (sizeof (PMpixeltype)) so buf must be 8 times npix.

To map from *screen space* to *subscreen-independent* processor space the coordinate conversion macros (PMilo, PMihi, PMjlo, PMjhi), etc. should be used with the global PMrealscrn subscreen pointer.

### NOTES

Subscreen-independent space is only an abstraction on top of subscreens. Although these routines do not use PMsubscrn pointers, they read and write pixels using subscreens, alternating between subscreens when needed. In most cases using the subscreen oriented routines will be faster because pixels are accessed linearly.

Refer to PMzbrk(3X) for page registers used.

### SEE ALSO

PMgetscan(3X) PMputscan(3X)

**PMgetscan** – read a scanline from a subscreen

## **SYNOPSIS**

#include <pxm.h>

void PMgetscan(scrn, buf, row, col, npix) PMsubscrn \*scrn; PMpixeltype \*buf; short row, col, npix;

#### DESCRIPTION

**PMgetscan** reads a row of *npix* pixels starting at (*col*, *row*) in subscreen *scrn* into the buffer *buf*, which must be large enough to hold the pixels.

scrn is a pointer to an initialized subscreen pointer. col and row are subscreen coordinates in the following legal ranges:

col [0,PMimax]

row [0,PMjmax]

*PMimax* and *PMjmax* are set to the appropriate value for the current model by system initialization (see the DEVtools *User's Guide* for more information on subscreen ranges).

Values beyond these ranges can generate unpredictable results.

SEE ALSO

PMgetcol(3X) PMgetrow(3X) PMputscan(3X)

PMgetzaddr - load a page register and return an address to a section of DRAM

### SYNOPSIS

#include <pxm.h>

char \*PMgetzaddr(desc)
PMzdesc desc;

### DESCRIPTION

**PMgetzaddr()** is called to gain access to the portion of DRAM memory allocated by **PMgetzdesc** via a pointer and page register. *desc* is the Z memory descriptor returned from a previous call to **PMgetzdesc**.

A table of available page registers is maintained by PMgetzaddr. Page registers 0 through 13 are available by default. Registers may be blocked by calls to the macros PMblock\_reg(), PMavail\_reg(), PMset\_lowreg() and PMset\_hireg(). The table is searched to see if the 1K row containing the memory to be accessed has been loaded into a page register. If the row has already been loaded, the number of accesses using that page register is incremented and the address is returned. If the row is not already loaded, an unaccessed page register is searched for and loaded with the page descriptor, if such a page register is found. The number of accesses to the page register is then incremented.

If no page registers are available, it will be necessary to call PMfreezaddr to free one up and temporarily restrict access to that block. By careful use of PMfreezaddr and PMgetzaddr and knowing how many page registers are available, it should be possible to never run out of page registers.

#### RETURNS

**PMgetzaddr**() returns a pointer to the valid memory address, if a page register can be found. NULL is returned on failure.

### NOTES

If **PMfreezaddr**() is called with the **PMzdesc** returned by **PMgetzaddr**(), and **PMgetzaddr**() is called again with the same **PMzdesc**, the value of the returned pointer may change, but the contents of the memory pointed to will not be changed.

Unpredictable results can occur if the memory past the end of the allocated block is accessed.

### SEE ALSO

PMfreezaddr(3X) PMgetzdesc(3X) PMzbrk(3X) PMblock\_reg(3X) PMavail\_reg(3X) PMset\_lowreg(3X) PMset\_hireg(3X)

PMgetzbuf - read a float value from the Z buffer

**SYNOPSIS** 

#include <pxm.h>

float \*PMgetzbuf(scrn, i, j, zptr)
PMsubscrn \*scrn;
short i, j;
float \*zptr;

# DESCRIPTION

**PMgetzbuf** reads a single value from Z buffer memory. *scrn* is a pointer to an initialized PMsubscrn structure corresponding to the subscreen from which the value is to be read.

*i* and *j* are subscreen coordinates with the following legal ranges:

i [0, PMimax]

j [0, PMjmax]

**PMimax** and **PMjmax** are set to the appropriate value for the current model by system initialization (see the *DEVtools User's Guide* for more information on subscreen ranges).

Values beyond these ranges will generate unpredictable results.

*zptr* is a pointer to a floating point number to be written with the Z value contained at (i, j) in *scrn*.

**PMgetzbuf** returns a pointer to the next Z value on the given row (i+1,j). This pointer can be used by **PMqzget** for more efficient Z buffer access.

For even faster access, the pointer returned can be used directly (unlike the pointer returned from **PMgetpix**) because Z buffer memory is fully mapped.

### NOTES

Refer to PMzbrk(3X) for page register use.

The pointer returned can be cast to other types to allow the Z memory to be used for *char*, *int* and other data types.

## EXAMPLE

PTR=PMgetzbuf(scrn,i,j,zval
 z2=ptr++
 z3=ptr++

### SEE ALSO

PMgetpix(3X) PMputzbuf(3X) PMqzget(3X) PMzget(3X)

PMgetzdesc, PMzdesc\_valid - allocate a DRAM block

### **SYNOPSIS**

#include <pxm.h>

PMzdesc PMgetzdesc(numbytes) int numbytes;

PMzdesc\_valid(desc) PMzdesc desc;

### DESCRIPTION

**PMgetzdesc()** is called after **PMzbrk()** has reserved the DRAM memory resources to allocate memory in blocks up to 1024 bytes. *numbytes* is the requested number of bytes, which must be less than or equal to 1024. The allocated memory is aligned on 4 byte boundaries.

**PMgetzdesc** returns a memory descriptor, of type PMzdesc, that contains two elements of addressing information. One element contains the number of the 1K block that holds the first available memory, and the other contains the offset of that memory from the beginning of that block. The offset is given in units of 4 bytes.

Memory is allocated from the beginning of the section reserved by **PMzbrk**() until the end of DRAM. No block may wrap over a 1K boundary, therefore, **PMgetzdesk** may have to skip over memory to guarantee this. Because of this, it is advisable to allocate memory in chunks that divide into 1024 evenly. Once a block of memory is allocated with **PMgetzdesc** it cannot be freed, except by reinitializing with a call to **PMzbrk**, which then starts the allocation process from the beginning.

In order to actually gain access to the memory being allocated, the descriptor must be used in a subsequent call to **PMgetzaddr()**.

### RETURNS

If successful, **PMgetzdesc** returns a descriptor of type **PMzdesc**, as described above. If there is no more reserved DRAM left or if the portion left is smaller than the *numbytes* requested, both elements of the returned descriptor are zero. Validity of a descriptor can be tested with the macro **PMzdesc\_valid(desc)**, where *desc* is the descriptor being tested. The value is non-zero if the result is valid.

### NOTES

Requesting more than 1024 bytes can produce unpredictable results.

SEE ALSO

PMgetzaddr(3X) PMfreezaddr(3X) PMzbrk(3X)

PMhost\_exit - send a message to the host that signals the completion of a Pixel Machine program

SYNOPSIS

void PMhost exit()

### DESCRIPTION

**PMhost\_exit** sends a message to the host that causes the **DEVpoll\_nodes** function to return to the caller. This is usually used to signal the completion of a Pixel Machine program, but may also be used in other applications where the Pixel Machine may want to request that **DEVpoll\_nodes** return to the caller.

### NOTES

If devprint is running on the host, PMhost exit will cause it to terminate.

#### SEE ALSO

devprint(1) DEVpoll\_nodes(3H) DEVwait\_exit(3H)

PMieee dsp - convert IEEE float to DSP float

SYNOPSIS

#include <libmath.h>

float \*PMieee\_dsp(len, ptr)
int len;
float \*ptr;

# DESCRIPTION

The *len* floating point numbers in IEEE format stored at *ptr* are converted to DSP32 format. A pointer immediately following the end of the array (ptr+len) is returned.

### SEE ALSO

PMlong\_dsp(3M)

**PMihi** - map from screen space (xmax) to processor space (ihi)

**SYNOPSIS** 

#include <pxm.h>

int PMihi(scrn, x)
PMsubscrn \*scrn;
float x;

# DESCRIPTION

**PMihi** performs the mapping from screen space to processor space. The domain transformation that maps from Cartesian (x, y) screen space to (i, j) processor space is as follows:

$$i = \frac{1}{Nx} (x - Ox)$$
$$j = \frac{1}{Ny} (y - Oy)$$

where Nx and Ny are the numbers of processors in the x and y directions, respectively, and Ox and Oy are the x and y offsets into the processor array, respectively. PMihi converts a screen space coordinate x to a processor space coordinate i that will guarantee satisfying the condition :

 $i Nx + Ox \leq x$ 

This ensures that all i values generated will map to screen coordinates less than or equal to x. The i value is always used as the last valid pixel to be rendered by a processor.

### NOTES

PMihi is implemented as a macro.

SEE ALSO

DEVtools User's Guide PMilo(3X) PMjlo(3X) PMjhi(3X)
PMilo - map from screen space (xmin) to processor space (ilo)

SYNOPSIS

#include <pxm.h>

int PMilo(scrn, x)
PMsubscrn \*scrn;
float x;

# DESCRIPTION

This macro performs the mapping from screen space to processor space. The domain transformation that maps from Cartesian (x,y) screen space to (i,j) processor space is as follows:

$$i = \frac{1}{Nx} (x - Ox)$$
$$j = \frac{1}{Ny} (y - Oy)$$

where Nx and Ny are the numbers of processors in the x and y directions, respectively, and Ox and Oy are the x and y offsets into the processor array, respectively.

**PMilo** converts a screen space coordinate x to a processor space coordinate i that guarantees satisfying the condition :

 $i Nx + Ox \ge x$ 

This ensures that all i values generated will map to screen coordinates greater than or equal to x. The i value is always used as the first valid pixel to be rendered by a processor.

### NOTES

PMilo is implemented as a macro.

SEE ALSO

DEVtools User's Guide PMihi(3X) PMjlo(3X) PMjhi(3X)

**PMint\_color** - macro that converts an integer to an internal color value

### SYNOPSIS

#include <pxm.h>

int PMint\_color(i)
int i;

## DESCRIPTION

**PMint\_color** is a macro that converts an integer in the range 0 - 255 to an internal color value. Results for input values outside of the supported range are undefined.

### SEE ALSO

PMcolor\_int(3N) PMcolor\_float(3N) PMfloat\_color(3N)

~ - -

**PMinterleave** – interleave or deinterleave a block

### SYNOPSIS

#include <pxm.h>
#include <sysmsg.h>

void PMinterleave(mode, dir, x, y, nx, ny, ram )

- int mode;
- int dir;
- int x, y;
- int nx, ny;
- int ram;

## DESCRIPTION

**PMinterleave()** deinterleaves or interleaves a rectangular region of the screen starting at (x,y) in screen space, for a size of nx pixels by ny scanlines in one dimension. The values of x and y are restricted to multiples of the number of processors in the x and y directions (*PMnx*, *PMny*), respectively.

nx and ny must be multiples of PMnx squared and PMny squared, respectively.

mode is either PM\_INTERLEAVE or PM\_DEINTERLEAVE, and specifies if this is an interleave or deinterleave operation.

dir is the dimension, either PM ROW INT or PM COL INT for horizontal or vertical.

x and y are the upper left hand coordinate of the block in screen space and are in the range [0-(PMxmax-1)] and [0-(PMymax-1)].

nx and ny are the number of pixels in the x and y direction, respectively, and are in the range [0-PMxmax)] and [0-PMxmax)].

The *ram* parameter is one of:

- PM VRAM1 BUFFER: uses VRAM1 instead of VRAM0 on a 932 and higher.
  - If in double buffer mode, (i,j) must be within the correct limits, otherwise they can be larger as with PM FRONT BUFFER.
- PM\_BACK\_BUFFER: the currently non-displayed buffer.
- PM\_FRONT\_BUFFER: the currently displayed buffer. Note, however, that in an appropriately large model in single buffer mode, you can specify i,j out of bounds, e.g., on a 964 (512,512) will work.

PM ZRAM BUFFER: uses ZRAM without subscreens.

To interleave (deinterleave) in two dimensions call **PMinterleave**() twice with the same parameters except change *dir* from PM ROW to PM COL (or vice-versa).

### NOTES

For PMinterleave() to work, the Pixel Machine must be equipped with the necessary SIO hardware.

This function changes the SIO direction. The host must be polling via a call to **DEVpoll\_nodes()** or running the **devprint(1)** utility. **PMpsync()** is called internally.

PMinterleave() needs 4200 bytes available on the stack.

Saves and restores any page registers that it uses.

SEE ALSO

PMpsync(3X) PMsiodir(3X) PMmsg\_exchange(3X) PMmsg\_setup(3X) PMsioinit(3X) DEVpoll\_nodes(3S) devprint(1)

2

**PMjhi** – map from screen space (ymax) to processor space (jhi)

### **SYNOPSIS**

#include <pxm.h>

int PMjhi(scrn, y)
PMsubscrn \*scrn;
float y;

#### DESCRIPTION

**PMjhi** performs the mapping from screen space to processor space. The domain transformation that maps from Cartesian (x,y) screen space to (i,j) processor space is as follows:

$$i = \frac{1}{Nx} (x - Ox)$$
$$j = \frac{1}{Ny} (y - Oy)$$

where Nx and Ny are the numbers of processors in the x and y directions, respectively, and Ox and Oy are the x and y offsets into the processor array, respectively.

**PMjhi** converts a screen space coordinate y to a processor space coordinate j that will guarantee satisfying the condition :

 $j Ny + Oy \leq y$ 

This ensures that all j values generated will map to screen coordinates less than or equal to y. The j value is always used as the last valid pixel to be rendered by a processor.

### NOTES

**PMjhi** is implemented as a macro.

SEE ALSO

DEVtools User's Guide PMilo(3X) PMihi(3X) PMjlo(3X)

PMjlo - map from screen space (ymin) to processor space (jlo)

SYNOPSIS

#include <pxm.h>

int PMjlo(scrn, y) PMsubscrn scrn; float y;

## DESCRIPTION

**PMjlo** performs the mapping from screen space to processor space. The domain transformation that maps from cartesian (x,y) screen space to (i,j) processor space is as follows:

$$i = \frac{1}{Nx} (x - Ox)$$
$$j = \frac{1}{Ny} (y - Oy)$$

where Nx and Ny are the numbers of processors in the x and y directions, respectively, and Ox and Oy are the x and y offsets into the processor array, respectively.

**PMjlo** converts a screen space coordinate y to a processor space coordinate j that will guarantee satisfying the condition :

$$j Ny + Oy \ge y$$

This ensures that all j values generated will map to screen coordinates greater than or equal to y. The j value is always used as the first valid pixel to be rendered by a processor.

# NOTES

PMjlo is implemented as a macro.

SEE ALSO

DEVtools User's Guide PMilo(3X) PMihi(3X) PMjhi(3X)

PMIdot - specialized dot product for light sources

SYNOPSIS

#include <libmath.h>

float PMldot(v0, v1) float v0[3], v1[3];

# DESCRIPTION

**PMIdot** calculates the dot product of vectors v0 and v1. If the result is negative, **PMIdot** returns zero, otherwise it returns the value of the dot product.

**PMlong\_dsp** - convert an array of longs to float

# SYNOPSIS

#include <libmath.h>

long \*PMlong\_dsp(len, ptr)
int len;
float \*ptr;

# DESCRIPTION

The len long numbers stored at ptr are converted to float. A pointer immediately following the end of the array (ptr+len) is returned.

# SEE ALSO

PMieee\_dsp(3M)

PMmsg exchange - send and receive data packet over serial links

**SYNOPSIS** 

#include <pxm.h>

void PMmsg\_exchange(inbuf, outbuf, length)
float \*inbuf, \*outbuf;
int length;

# DESCRIPTION

**PMmsg\_exchange** sends *length* floats from *outbuf* out the serial link, then waits to receive *length* floats into *inbuf* on the link. Because of restrictions imposed by hardware, all nodes must exchange the same amount of data at the same time; the correct procedure to do this uses the **PMmsg\_setup** and **PMpsync** routines as follows:

```
float inbuf[SIZE], outbuf[SIZE];
PMmsg_setup(inbuf);
PMpsync();
PMmsg exchange(inbuf, outbuf, SIZE);
```

Any data type may be exchanged over the link, but the packet size must be a multiple of 4 bytes (sizeof(float)).

### NOTES

The *inbuf* pointers passed to PMmsg\_setup and PMmsg\_exchange must be the same or PMmsg exchange may never return.

PMsioinit must be called before any other use of the serial links is made.

### SEE ALSO

PMmsg\_setup(3X) PMpsync(3X) PMsioinit(3X)

PMmsg setup - set serial DMA input pointer

# SYNOPSIS

#include <pxm.h>

void PMmsg\_setup(buffer)
float \*buffer;

## DESCRIPTION

**PMmsg\_setup** sets the serial DMA input pointer to the supplied *buffer*. The pointer must be set and all processors synchronized using **PMpsync** before **PMmsg\_exchange** functions correctly.

SEE ALSO

PMmsg\_exchange(3X) PMpsync(3X) PMsioinit(3X)

PMmyx - test if a given screen space coordinate is in processor space

## SYNOPSIS

#include <pxm.h>

int PMmyx(scrn, x)
PMsubscrn \*scrn;
float x;

# DESCRIPTION

**PMmyx** tests if the screen space coordinate x is in the processor subscreen *scrn* and returns TRUE or FALSE accordingly.

### NOTES

PMmyx is implemented as a macro.

SEE ALSO

PMmyy(3X) PMxat(3X) PMyat(3X)

PMmyy - test if a given screen space coordinate is in processor space

# SYNOPSIS

#include <pxm.h>

int PMmyy(scrn, y)
PMsubscrn \*scrn;
float y;

# DESCRIPTION

**PMmyy** tests if the screen space coordinate y is in the processor subscreen *scrn* and returns TRUE or FALSE accordingly.

# NOTES

PMmyy is implemented as a macro.

SEE ALSO

PMmyx(3X) PMxat(3X) PMyat(3X)

PMnorm - normalize a 3D vector and return its length

# SYNOPSIS

#include <libmath.h>

float PMnorm(v)
float v[3];

# DESCRIPTION

**PMnorm** normalizes the vector v, and overwrites v with this new value. It returns the inverse of the length of vector v prior to normalization.

.

**PMoutpir** – output a value to the PIR register

# SYNOPSIS

void PMoutpir(val)
short val;

## DESCRIPTION

**PMoutpir** waits until the PIR is empty and then writes *val* to it. The wait ensures that the host has read all values written with previous calls to **PMoutpir**.

This function is a low level I/O routine; most applications should use PMusermsg() instead.

## SEE ALSO

PMusermsg(3N)

PMoverlay - turn overlay on or off

## SYNOPSIS

void PMoverlay(flag)
int flag;

# DESCRIPTION

PMoverlay sets the overlay bit in the pixel node flag register to turn the overlay capability on or off.

If flag is zero, overlay is disabled (the default). A nonzero value for flag turns overlay on.

# NOTES

In addition to calling **PMoverlay**, **DEVpixel\_mode\_overlay** must also be called on the host to set the desired overlay mode.

# SEE ALSO

DEVpixel\_mode\_overlay(3S)

PMpagereg, PMdesc, PMxlate - macros to manipulate page registers used to access video and Z memory

#### **SYNOPSIS**

#include <pxm.h>
#include <pixel.h>

int PMpagereg(reg\_number) int reg number;

```
int PMdesc(bank, mode)
int bank;
int mode;
```

int PMxlate(reg\_number)
int reg number;

#### DESCRIPTION

These macros are used to manipulate the page registers used to access the video memory and Z memory.

The page registers are located in a reserved memory area. The **PMpagereg** macro is used to generate the address of a specified page register. *reg\_number* is the number of the register whose address is to be supplied and is in the range [0-15].

The **PMdesc** macro is used to generate the value to be stored into a page register in order to access a given bank of memory. *bank* designates the bank of memory to be accessed and must be one of:

PM_ZMEM	– Z memory
PM_RG0	<ul> <li>red/green bank of VRAM0</li> </ul>
PM_BO0	- blue/overlay bank of VRAM0
PM <sup>-</sup> RG1	<ul> <li>red/green bank of VRAM1</li> </ul>
PM <sup>BO1</sup>	- blue/overlay bank of VRAM1

*mode* must be either PM\_FIX\_ROW or PM\_FIX\_COL; PM\_FIX\_ROW is used to access the pixels of a given scan line. PM\_FIX\_COL is used to access the pixels of a given column. The row number (in fixed row mode) or column number (in fixed column mode) is added to the value returned by PMdesc to create the descriptor needed to access the desired memory row or column.

PMxlate generates a pointer than can be used to access the contents of the row or column specified by the **PMdesc** macro. Once a page register has been established, the next 1024 bytes can be accessed using the pointer generated by the **PMxlate** macro.

### **EXAMPLE**

The following is an example of these macros. This programs turns on all of the red pixels in VRAM0 and the blue pixels in VRAM1, and turns off the green pixels in VRAM0 and the overlay pixels in VRAM1.

#include <pxm.h>
#include <pixel.h>
#define RGREG 6
#define BOREG 7

```
main()
ł
      register int
                        i;
      register int
                        j;
      register int
                        *rgptr;
                       *boptr;
      register int
      register int
                        *rgpagereg;
                        *bopagereg;
      register int
      rgpagereg = (int *)PMpagereg(RGREG);
      bopagereg = (int *)PMpagereg(BOREG);
      for (j = 0; j < 255; ++j) {
            *rgpagereg = PMdesc(PM RG0, PM FIX ROW) + j;
            *bopagereg = PMdesc(PM_BO1, PM_FIX_ROW) + j;
            rgptr = (int *)PMxlate(RGREG);
            boptr = (int *)PMxlate(BOREG);
            for (i = 0; i < 255; ++i) {
                  *rgptr++ = PMint_color(255); /* Set red */
                  *rgptr++ = PMint color(0); /* Clear green
                  *boptr++ = PMint color(255); /* Set blue */
                  *boptr++ = PMint color(0); /* Clear alpha
                                                                   */
            }
      }
}
```

NOTES

The **pixel.h** include file can be used with both C and assembler source files. As a result, the macro return values are not cast as pointers. For this reason, you must cast the return value of the macros to the appropriate pointer type.

**PMpagereg** should always be cast as a pointer to an *int*. **PMdesc** really does return an integer. **PMxlate** should be cast to an appropriate type based on the application. When dealing with VRAM (as opposed to Z memory), the pointer returned by **PMxlate** is usually a pointer to an *int*.

Some of the DEVtools pixel node functions set page registers automatically, and other functions rely on them. See PMzbrk for the list of page registers used.

Page registers 14 and 15 are reserved for use by the host for DMA.

SEE ALSO

PMzbrk(3X)

**PMpixaddr** – generate a pointer to a specific pixel

**SYNOPSIS** 

#include <pxm.h>

short \*PMpixaddr(scrn, i, j)
PMsubscrn \*scrn;
short i, j;

## DESCRIPTION

**PMpixaddr** generates addresses of pixels in the frame buffer. *scrn* is a pointer to an initialized PMsubscrn structure corresponding to the subscreen in which the desired pixel lies.

*i* and *j* are subscreen coordinates with the following legal ranges:

*i* [0, PMimax] *j* [0, PMjmax]

**PMimax** and **PMjmax** are set to the appropriate value for the current model by system initialization (see the *DEVtools User's Guide* for more information on subscreen ranges).

Values beyond these ranges will generate unpredictable results.

**PMpixaddr** returns a pointer to the pixel at coordinates (i, j) in subscreen *scrn*. This pointer can be used by **PMqget** and **PMqput** for more efficient frame buffer access.

### NOTES

Refer to PMzbrk(3X) for page register use.

#### SEE ALSO

PMgetpix(3X) PMputpix(3X) PMqget(3X) PMqput(3X)

PMpow - power function

# SYNOPSIS

#include <libmath.h>

float PMpow(x, y) float x, y;

DESCRIPTION

**PMpow** returns the quantity  $x \, y$ , where both x and y are floating point values. x should be of positive magnitude.

# SEE ALSO

PMx\_exp\_n(3M)

PMpsync – wait for all pixel processors to synchronize

# SYNOPSIS

void PMpsync()

# DESCRIPTION

**PMpsync** is a processor synchronization primitive. Once called, it will not return until *all* pixel nodes have called **PMpsync**.

# NOTES

PMpsync uses the PM\_FLAG hardware signal; thus PMflagled and PMpsync should not be used in the same program.

# SEE ALSO

PMvsync(3X)

PMputcmd - write opcode, parameter count, and parameters to the output FIFO of a pipe node

# SYNOPSIS

#include <pxm.h>

void PMputcmd()

# DESCRIPTION

**PMputcmd** copies the opcode, count, and parameters from the global **PMcommand** structure to the output FIFO.

## NOTES

PMputcmd can only be called from a pipe node program.

## SEE ALSO

PMcommand(4N) PMgetdata(3P) PMgetop(3P) PMputdata(3P) PMputop(3P)

PMputdata - write parameters to the output FIFO of a pipe node

# SYNOPSIS

#include <pxm.h>

# void PMputdata()

## DESCRIPTION

PMputdata copies the parameters from the global PMcommand structure to the output FIFO.

# NOTES

PMputdata can only be called from a pipe node program.

PMputdata must be preceded by a call to PMputop.

### SEE ALSO

PMcommand(4N) PMgetdata(3P) PMgetop(3P) PMputcmd(3P) PMputop(3P)

PMputop - write opcode and parameter count to the output FIFO of a pipe node

# SYNOPSIS

#include <pxm.h>

void PMputop()

# DESCRIPTION

**PMputop** copies the opcode and parameter count from the global **PMcommand** structure to the output FIFO.

## NOTES

**PMputop** can only be called from a pipe node program.

PMputop must be followed by a call to PMputdata if PMcommand.count is non-zero.

# SEE ALSO

PMcommand(4N) PMgetdata(3P) PMgetop(3P) PMputcmd(3P) PMputdata(3P)

**PMputpix** – output a pixel to the current buffer

## SYNOPSIS

#include <pxm.h>

short \*PMputpix(scrn, i, j, color)
PMsubscrn \*scrn;
short i, j;
PMpixeltype \*color;

### DESCRIPTION

**PMputpix** writes a single pixel to the frame buffer. *scrn* is a pointer to an initialized **PMsubscrn** structure corresponding to the subscreen to which the pixel is written.

*i* and *j* are subscreen coordinates with the following legal ranges:

*i* [0, PMimax]

*j* [0, PMjmax]

**PMimax** and **PMjmax** are set to the appropriate value for the current model by system initialization (see the *DEVtools User's Guide* for more information on subscreen ranges).

Values beyond these ranges will generate unpredictable results.

color is a pointer to a PMpixeltype structure whose red, green, blue and overlay components are written at (i, j) in scrn.

**PMputpix** returns a pointer to the next pixel on the given row (i+1,j). This pointer may be used by **PMqput** for more efficient frame buffer access.

# NOTES

Refer to PMzbrk(3X) for page register use.

### SEE ALSO

PMgetpix(3X) PMqput(3X)

**PMputscan** – write a scanline to a subscreen

# SYNOPSIS

#include <pxm.h>

void PMputscan(scrn, buf, row, col, npix) PMsubscrn \*scrn; PMpixeltype \*buf; short row, col, npix;

## DESCRIPTION

PMputscan writes a row of npix pixels starting at (col, row) in subscreen scrn from the buffer buf.

scrn is a pointer to an initialized subscreen pointer. col and row are subscreen coordinates in the following legal ranges:

col [0,PMimax]

row [0,PMjmax]

*PMimax* and *PMjmax* are set to the appropriate value for the current model by system initialization (see the DEVtools *User's Guide* for more information on subscreen ranges).

Values beyond these ranges can generate unpredictable results.

# SEE ALSO

PMgetscan(3X)

PMputzbuf - write a float value to the Z buffer

## SYNOPSIS

#include <pxm.h>

```
float *PMputzbuf( scrn, i, j, zval )
PMsubscrn *scrn;
short i, j;
float zval;
```

## DESCRIPTION

**PMputzbuf** writes a single value to Z buffer memory. *scrn* is a pointer to an initialized **PMsubscrn** structure corresponding to the subscreen from which the value is to be read.

*i* and *j* are subscreen coordinates with the following legal ranges:

- i [0, PMimax]
- j [0, PMjmax]

**PMimax** and **PMjmax** are set to the appropriate value for the current model by system initialization (see the *DEVtools User's Guide* for more information on subscreen ranges.

Values beyond these ranges will generate unpredictable results.

zval is a floating point value to be written at (i, j) in scrn.

**PMputzbuf** returns a pointer to the next Z value on the given row (i+1, j).

The pointer returned can be used directly (unlike the pointer returned from PMputpix), because Z buffer memory is fully mapped.

### NOTES

Refer to PMzbrk(3X) for page register use.

The pointer returned can be cast to other types to allow Z memory to be used for *char*, *int*, and other data types.

### EXAMPLE

```
ptr=PMputzbuf(scrn, i,j, zval);
*ptr++=zval;
*ptr++=zval;
```

### SEE ALSO

PMgetzbuf(3X) PMputpix(3X) PMzput(3X)

PMqcopyztoz - copy from one section of DRAM to another

# SYNOPSIS

#include <pxm.h>

void PMqcopyztoz(start\_i, start\_j, len\_i, len\_j, dest\_i, dest\_j)
int start\_i;
int start\_j;
int len\_i;
int len\_j;
int dest\_i;
int dest\_j;

## DESCRIPTION

**PMqcopyztoz** copies a rectangular section of DRAM, *len\_i* long words by *len\_j* rows, from coordinates *start\_i* and *start\_j* to another section of DRAM buffer. *dest\_i* and *dest\_j* are the destination coordinates, and correspond to *start\_i* and *start\_j*, respectively. **PMqcopyztoz** is faster and takes less code space than **PMcopyztoz(3)**, but cannot handle overlapping copies. While some overlapping copies may succeed, care should be taken so that the source area and destination areas of ZRAM are disjoint.

The \_i arguments are in units of 4 byte long words, e.g., 1 byte for each of red, green, blue and overlay, or the size of one float. Thus, if *start\_i* is set to 1, and *len\_i* is set to 1, 4 bytes will be copied on each row, starting at an offset of 4 bytes from the beginning of the row. In the \_j direction, one row corresponds to one row, with no multiplicative factors.

## SEE ALSO

PMcopyztoz(3X) PMcopyztov(3X) PMcopyvtoz(3X)

PMqget - quick read of a pixel from the current buffer

# SYNOPSIS

#include <pxm.h>

short \*PMqget(color, ptr)
PMpixeltype \*color;
short \*ptr;

# DESCRIPTION

**PMqget** reads a single pixel from the frame buffer. *ptr* is a pointer to the pixel location from which the pixel is to be read; *color* is a pointer to a PMpixeltype structure which is written with the pixel located at *ptr*.

PMqget returns a pointer to the next pixel on the given row. This value may be used in subsequent calls to PMqget.

## NOTES

PMqget uses a pointer created by PMgetpix, PMv0get and other routines. PMqget uses the same page registers as the routine that generated the pointer. The user must ensure that the page registers are not corrupted while PMqget is in use.

Refer to PMzbrk(3X) for page register use.

## SEE ALSO

PMgetpix(3X) PMpixaddr(3X) PMqput(3X)

PMqput - quick write of a pixel to the current buffer

# SYNOPSIS

#include <pxm.h>

short \*PMqput(color, ptr)
PMpixeltype \*color;
short \*ptr;

## DESCRIPTION

**PMqput** writes a single pixel from the frame buffer. *ptr* is a pointer to the pixel location to which the pixel is to be written; *color* is a pointer to a PMpixeltype structure containing the pixel to be written at *ptr*.

**PMqput** returns a pointer to the next pixel on the given row. This value may be used in subsequent calls to **PMqput**.

# NOTES

**PMqput** uses a pointer created by **PMputpix**, **PMv0put** and other routines. **PMqput** uses the same page registers as the routine that generated the pointer. The user must ensure that the page registers are not corrupted while **PMqput** is in use.

Refer to PMzbrk(3X) for page register use.

### SEE ALSO

PMpixaddr(3X) PMputpix(3X) PMqget(3X)

**PMrdyled** - turn the PM\_RDY LED on or off

# SYNOPSIS

#include <pxm.h>

void PMrdyled(flag)
short flag;

## DESCRIPTION

**PMrdyled** clears (if flag == 0) or sets (if flag != 0) the PM\_RDY LED for this node.

# NOTES

PMrdyled uses the PM\_RDY hardware signal; thus PMrdyled and PMvsync should not be used in the same program.

# SEE ALSO

PMflagled(3X) PMvsync(3X)

PMrdyoff - turn the ready signal off

# SYNOPSIS

void PMrdyoff()

# DESCRIPTION

**PMrdyoff** turns off the DEV\_FLAG signal used by **PMvsync**. It must be called some time after calling **PMvsync** and before another **PMvsync** is done.

# NOTES

The purpose of separating PMvsync and PMrdyoff is to allow as much time as possible for user code after vertical retrace begins.

# SEE ALSO

PMvsync(3X)

PMsetsem - set the semaphore

# SYNOPSIS

void PMsetsem(value)
short value;

# DESCRIPTION

PMsetsem waits for the software semaphore to be cleared by the host, then sets it to the passed value.

### SEE ALSO

PMwaitsem(3N)

.

PMsin - trigonometric function

# SYNOPSIS

#include <libmath.h>

float PMsin(theta) float theta;

# DESCRIPTION

PMsin returns the sine of *theta*.

theta must be in radians and be between -Pi/2 and +Pi/2.

PMsiodir - set serial I/O link direction

### **SYNOPSIS**

#include <sysmsg.h>

void PMsiodir(dir) short dir;

### DESCRIPTION

PMsiodir sends a message to the host monitor process to set the serial I/O (SIO) link direction. dir must be one of:

PM\_MSG\_SERIAL\_NORTH PM\_MSG\_SERIAL\_SOUTH PM\_MSG\_SERIAL\_EAST PM\_MSG\_SERIAL\_WEST

These constants are defined in sysmsg.h.

For it to work correctly, all the pixel nodes must call **PMsiodir**. **PMsiodir** calls **PMpsync** internally to synchronize before the host changes the link direction for all the pixel nodes.

# NOTES

As with all other SIO functions, PMsiodir must only be called from pixel nodes.

### SEE ALSO

PMmsg\_exchange(3X) PMmsg\_setup(3X) PMpsync(3X) PMsioinit(3X)

PMsioinit - initialize serial I/O

# SYNOPSIS

#include <pxm.h>

# void PMsioinit()

# DESCRIPTION

PMsioinit configures the serial I/O link for DMA input and polled output. It must be called only once before attempting to send messages over the serial links using PMmsg\_setup() and PMmsg\_exchange().

# SEE ALSO

PMmsg\_setup(3X) PMmsg\_exchange(3X) PMpsync(3X)

PMsnglbuff - disable double buffering mode

## SYNOPSIS

#include <pxm.h>

# void PMsnglbuff()

## DESCRIPTION

**PMsnglbuff** disables double buffering and returns to single buffer mode. This means that all future updates using subscreen oriented functions (e.g., **PMputpix()**) will occur in the same buffer that is displayed.

PMsnglbuff only needs to be called after a call to PMdblbuff because it is the default mode at start up.

## SEE ALSO

PMswapbuff(3X) PMswapback(3X) PMdblbuff(3X)

,
PMsqrt - sqare root function

# SYNOPSIS

#include <libmath.h>

# float PMsqrt(x) float x;

# DESCRIPTION

**PMsqrt** returns the square root of x. x must be  $\geq 0$ . This function is accurate to 6 significant digits.

**PMswap\_pipe** – switch primary and alternate pipes of a dual pipe system **PMbus wait** – wait until control of the broadcast bus is granted

## SYNOPSIS

#include <pxm.h>

void PMswap\_pipe()
void PMbus wait()

#### DESCRIPTION

On a dual pipe system, with the pipes operating in parallel mode, one pipe is the primary pipe and the other is the alternate pipe. **PMswap\_pipe** reverses the functions of the two pipes. This is used to balance the load between the two pipes.

**PMswap\_pipe** can only be called by the last node of a pipe board (node 8 or 17). When called, **PMswap\_pipe** releases the broadcast bus to the alternate pipe It then requests the bus and waits for bus access to be granted.

**PMbus\_wait** loops until control of the bus is granted. This is typically called during the initialization phase by the second pipe board, because initial control of the pipe is granted to the first pipe board.

#### SEE ALSO

DEVswap\_pipe(3P)

PMswapback - swap meaning of back buffer

# SYNOPSIS

#include <pxm.h>

void PMswapback( )

# DESCRIPTION

**PMswapback** swaps the back and front buffer with respect to update, but does not change the visible buffer. In double buffer mode this means that the front buffer is also the update the buffer. In single buffer mode this means that the back buffer is updated. This functions will change the behavior of all functions that use a PMsubscrn argument to update the current buffer.

# NOTES

PMswapback is implemented as a macro.

PMswapbuff - swap front and back pixel buffers

# SYNOPSIS

void PMswapbuff()

# DESCRIPTION

**PMswapbuff** exchanges the front (visible) and back pixel buffers; it should be called when a frame has been generated in the pixel buffer and must be displayed. **PMswapbuff** waits for vertical retrace by calling **PMvsync** before swapping and then calling **PMrdyoff**.

## NOTES

Double buffering mode must be enabled with PMdblbuff before calling PMswapbuff.

SEE ALSO

PMdblbuff(3X) PMswapback(3X) PMsnglbuff(3X) PMvsync(3X) PMrdyoff(3X)

**PMusermsg** – send a *user message* to the host

#### SYNOPSIS

void PMusermsg(msg)
short msg;

# DESCRIPTION

**PMusermsg()** sends a user defined opcode (a *user message*) to the host monitor process. *msg* must be a positive short int.

**PMusermsg()** checks the software semaphore to see if there was a previous **PMusermsg()** pending and, if necessary, waits. Otherwise, **PMusermsg()** returns immediately. If the message operation must complete before execution continues, **PMwaitsem()** should be called.

# SEE ALSO

PMwaitsem(3N) DEVtoolsUser's Guide (section on user messages)

PMv0get - read a pixel from buffer 0

# SYNOPSIS

#include <pxm.h>

short \*PMv0get(i, j, color)
short i, j;
PMpixeltype \*color;

# DESCRIPTION

**PMv0get()** reads a single pixel from the frame buffer. Unlike **PMgetpix()**, the coordinate system used allows full access to frame buffer memory.

i and j are coordinates in the range [0, 255]. Values outside this range will generate unpredictable results.

*color* is a pointer to a PMpixeltype structure whose red, green, blue and overlay components are loaded with the pixel data contained at (i, j) in page 0 of pixel memory.

**PMv0get()** returns a pointer to the next pixel on the given row (i+1,j). This pointer can be used by **PMqget()** for more efficient frame buffer access.

## NOTES

Refer to PMzbrk(3X) for page register use.

PMv0get() does not take into account subscreens or front and back buffers.

SEE ALSO

PMgetpix(3X) PMqget(3X) PMv0put(3X) PMv1get(3X)

**PMv0put** – write a pixel to buffer 0

## SYNOPSIS

#include <pxm.h>

short \*PMv0put(i, j, color)
short i, j;
PMpixeltype \*color;

#### DESCRIPTION

**PMv0put()** writes a single pixel to the frame buffer. Unlike **PMputpix()**, the coordinate system used allows full access to frame buffer memory.

i and j are coordinates in the range [0, 255]. Values outside this range will generate unpredictable results.

*color* is a pointer to a PMpixeltype structure which contains the red, green, blue and overlay components to be written at (i, j) in page 0 of pixel memory.

**PMv0put()** returns a pointer to the next pixel on the given row (i+1,j). This pointer can be used by **PMqput()** for more efficient frame buffer access.

#### NOTES

Refer to PMzbrk(3X) for page register use.

PMv0put() does not take into account subscreens or front and back buffers.

SEE ALSO

PMputpix(3X) PMqput(3X) PMv0get(3X) PMv1put(3X)

PMv1get - read a pixel from buffer 1

## SYNOPSIS

#include <pxm.h>

short \*PMv1get(i, j, color)
short i, j;
PMpixeltype \*color;

# DESCRIPTION

**PMv1get()** reads a single pixel from the frame buffer. Unlike **PMgetpix()**, the coordinate system used allows full access to frame buffer memory.

i and j are coordinates in the range [0, 255]. Values outside this range will generate unpredictable results.

*color* is a pointer to a PMpixeltype structure whose red, green, blue and overlay components will be loaded with the pixel data contained at (i, j) in page 1 of pixel memory.

**PMv1get()** returns a pointer to the next pixel on the given row (i+1,j). This pointer can be used by **PMqget()** for more efficient frame buffer access.

## NOTES

Refer to PMzbrk(3X) for page register use.

PMv1get() does not take into account subscreens or front and back buffers.

SEE ALSO

PMgetpix(3X) PMqget(3X) PMv1put(3X) PMv0get(3X)

**PMv1put** - write a pixel to buffer 1

# **SYNOPSIS**

#include <pxm.h>

short \*PMv1put(i, j, color)
short i, j;
PMpixeltype \*color;

# DESCRIPTION

**PMv1put()** writes a single pixel to the frame buffer. Unlike **PMputpix()**, the coordinate system used allows full access to frame buffer memory.

i and j are coordinates in the range [0, 255]. Values outside this range will generate unpredictable results.

*color* is a pointer to a PMpixeltype structure which contains the red, green, blue and overlay components to be written at (i, j) in page 1 of pixel memory.

**PMv1put()** returns a pointer to the next pixel on the given row (i+1,j). This pointer can be used by **PMqput()** for more efficient frame buffer access.

# NOTES

Refer to PMzbrk(3X) for page register use.

PMv1put() does not take into account subscreens or front and back buffers.

SEE ALSO

PMputpix(3X) PMqput(3X) PMv1get(3X) PMv0put(3X)

PMvsync – synchronize and wait for vertical retrace

# SYNOPSIS

void PMvsync()

# DESCRIPTION

**PMvsync** is a video synchronization primitive. Once called, it will not return until both of the following conditions are true:

all pixel nodes have called PMvsync.

the vertical retrace period has begun.

#### NOTES

**PMvsync** uses the PM\_RDY hardware signal; thus **PMrdyled** and **PMvsync** should not be used in the same program.

**PMrdyoff** *must* be called after calling **PMvsync** and before further calls to **PMvsync** are made. The purpose of separating **PMvsync** and **PMrdyoff** is to allow as much time as possible for user code after vertical retrace begins.

PMswapbuff uses PMvsync and PMrdyoff internally.

# SEE ALSO

PMpsync(3X) PMrdyoff(3X) PMswapbuff(3X)

PMwaitsem - wait for semaphore to clear

# SYNOPSIS

void PMwaitsem()

# DESCRIPTION

**PMwaitsem** polls the software semaphore until it is cleared by the host. It can be used to synchronize with the host after calling **PMsetsem**, or to wait for the host to complete a *user message* or *system message* such as **printf**.

# SEE ALSO

PMsetsem(3N) PMusermsg(3N) printf(3N) DEVtools*User's Guide* (section on messages)

PMx\_exp\_n – integer power function

SYNOPSIS

#include <libmath.h>

float PMx\_exp\_n(x, n)
float x;
short n;

# DESCRIPTION

**PMx\_exp\_n** returns the quantity x n, where *n* is a positive integer between 1 and 20.

# SEE ALSO

PMpow(3M)

PMxat - map subscreen coordinates to screen space

SYNOPSIS

#include <pxm.h>

float PMxat(scrn, i) PMsubscrn \*scrn; short i;

# DESCRIPTION

**PMxat** maps the subscreen coordinate i to the corresponding screen space x.

The mappings are:

$$x = i N_x + O_x$$

 $y = j N_y + O_y$ 

where  $N_x$  and  $N_y$  are the numbers of processors in the x and y directions, respectively, and  $O_x$  and  $O_y$  are the x and y offsets into the processor array, respectively.

NOTES

PMxat is implemented as a macro.

SEE ALSO

PMmyx(3X) PMmyy(3X) PMyat(3X)

PMyat - map subscreen coordinates to screen space

SYNOPSIS

#include <pxm.h>

float PMyat(scrn, j) PMsubscrn \*scrn; short j;

#### DESCRIPTION

**PMyat** maps the subscreen coordinate j to the corresponding screen space y.

The mappings are:

$$x = i N_x + O_x$$
$$y = j N_y + O_y$$

where  $N_x$  and  $N_y$  are the numbers of processors in the x and y directions, respectively, and  $O_x$  and  $O_y$  are the x and y offsets into the processor array, respectively.

NOTES

PMyat is implemented as a macro.

SEE ALSO

PMmyx(3X) PMmyy(3X) PMxat(3X)

**PMzaddr** – generate a ZRAM pointer to a row

SYNOPSIS

#include <pxm.h>

float \*PMzaddr(i, j)
int i, j;

# DESCRIPTION

**PMzaddr()** loads a page register with an appropriate descriptor, and then constructs a valid pointer that references that page register.

**PMzaddr** returns a pointer to the z value on the given row (i, j). Because the page register is loaded in fixed row addressing mode, the pointer can be used directly up to the end of the given row. To generate a column mode address use **PMzaddrcol**().

i and j are coordinates in the range [0, 255]. Values outside this range will generate unpredictable results.

# NOTES

Refer to PMzbrk(3X) for page register use.

PMzaddr() does not consider the PMsubscrn structure.

The pointer returned by this function can be cast to allow access to *char*, *int*, or other types of data stored in Z memory.

SEE ALSO

PMgetzbuf(3X) PMzaddrcol(3X) PMqzput(3X) PMqzget(3X) PMv0get(3X)

PMzaddrcol - generate a ZRAM pointer to a column

## SYNOPSIS

#include <pxm.h>

float \*PMzaddrcol(i, j)
int i, j;

# DESCRIPTION

**PMzaddrcol()** loads a page register with an appropriate descriptor, and then constructs a valid pointer that references that page register.

**PMzaddrcol**() returns a pointer to the z value on the given row (i, j). Because the page register is loaded in fixed column addressing mode, the pointer can be used directly up to the end of the given column. To generate a row mode address use **PMzaddr**().

i and j are coordinates in the range [0, 255]. Values outside this range will generate unpredictable results.

# NOTES

Refer to PMzbrk(3X) for page register use.

PMzaddrcol() does not consider the PMsubscrn structure.

The pointer returned by this function can be cast to allow access to *char*, *int*, or other types of data stored in Z memory.

SEE ALSO

PMgetzbuf(3X) PMzaddr(3X) PMqzput(3X) PMqzget(3X) PMv0get(3X)

PMzbrk, PMblock\_reg, PMavail\_reg, PMset\_lowreg, PMset\_hireg - reserve DRAM and page registers for dynamic allocation

#### SYNOPSIS

#include <pxm.h>

PMzdesc PMzbrk(numblocks) int numblocks;

#include <pageregs.h>

PMblock\_reg(n)
int n;

PMavail\_reg(n)
int n;

PMset\_lowreg(n)
int n;

PMset\_hireg(n) int n;

#### DESCRIPTION

**PMzbrk** is the initialization call to create a list of memory resources for DRAM (also called ZRAM) that are used in subsequent calls to **PMgetzaddr()**, **PMgetzdesc()** and **PMfreezaddr()**. *numblocks* is the number of kilobytes (or rows) of DRAM to reserve and is in the range of 1 to 256 inclusive. The memory is reserved from the end of DRAM. For example, **PMzbrk(2)** reserves the last 2 rows of DRAM, rows 254 and 255.

The macros **PMblock\_reg()**, **PMavail\_reg()**, **PMset\_lowreg()** and **PMset\_hireg()**, defined in *pageregs.h*, are provided as a way of manipulating the list of page registers that are made available to access DRAM through calls to **PMgetzaddr()**. By default **PMzbrk** makes the page registers in the range 0 to 13 inclusive, available. These macros only have an affect when called after **PMzbrk**.

**PMblock\_reg()** and **PMavail\_reg()** are used to specify individual page registers to be excluded or included, respectively, from use by **PMgetzaddr.** 

Another way to specify the page registers is to provide a range with calls to PMset\_lowreg() and PMset\_hireg(). The range is inclusive. For example, the calls

PMset\_lowreg(10) PMset\_hireg(13)

indicate that the page registers 10, 11, 12 and 13 can be used by **PMgetzaddr()**. The low and high registers can be set in the range from 0 to 13, inclusive. Page registers 14 and 15 are reserved for use by the host.

It is necessary to block certain page registers to avoid conflicts when using other DEVtools functions such as **PMgetpix**, that use page registers internally. If a page register is no longer needed by a specific routine later on, it could be made available to **PMgetzaddr** with a call to **PMavail reg()**.

Register assignments are given in the following table.

Page Register Assignments				
Page Register	Function			
0	PMgetscan(), PMputscan(), PMclear(), PMgetcol(), PMputcol(),			
_	PMgetrow(), PMputrow()			
1	PMgetscan(), PMputscan(), PMclear(), PMgetcol(), PMputcol(),			
	PMgetrow(), PMputrow()			
2	PMv0get(), PMgetpix(), PMgetcol(), PMputcol(), PMgetrow(), PMputrow()			
3	PMv0get(), PMgetpix(), PMgetcol(), PMputcol(), PMgetrow(), PMputrow()			
4	PMv0put(), PMputpix()			
5	PMv0put(), PMputpix()			
6	PMv1get()			
7	PMv1get()			
8	PMv1put()			
9	PMv1put()			
10	PMpixaddr()			
11	PMpixaddr()			
12	PMzget(), PMgetzbuf(), PMzaddr()			
13	PMzput(), PMputzbuf(), PMzaddrcol()			
14	Reserved for host use			
15	Reserved for host use			

# NOTES

Requesting a number of blocks greater than 256 can cause PMgetzdesc() to fail in unpredictable ways.

# SEE ALSO

PMgetzaddr(3X) PMgetzdesc(3X) PMfreezaddr(3X)

PMzget - read a float from the z buffer

SYNOPSIS

#include <pxm.h>

float \*PMzget(i, j, zptr)
short i, j;
float \*zptr;

# DESCRIPTION

**PMzget()** reads a single z value from the Z buffer. Unlike **PMgetzbuf()**, the coordinate system used allows full access to z buffer memory.

i and j are coordinates in the range [0, 255]. Values outside this range will generate unpredictable results.

*zptr* is a pointer to a floating point variable that will be written with the z value contained at (i, j) in the z buffer.

**PMzget()** returns a pointer to the next z value on the given row (i+1,j). This pointer can be used by **PMqzget()** for more efficient frame buffer access. For even faster access, the pointer returned can be used directly (unlike the pointer returned from **PMv0get()**) because z buffer memory is fully mapped.

#### NOTES

Refer to PMzbrk(3X) for page register use.

PMzget() does not consider subscreens.

The pointer returned by this function can be cast to allow access to *char*, *int*, or other types of data stored in Z memory.

SEE ALSO

PMgetzbuf(3X) PMqzget(3X) PMv0get(3X)

**PMzput** – write a float to the Z-buffer

**SYNOPSIS** 

#include <pxm.h>

float \*PMzput(i, j, zval) short i, j; float zval:

#### DESCRIPTION

**PMzput** writes a single Z value to the Z buffer. Unlike **PMputzbuf**, the coordinate system used allows full access to Z buffer memory.

i and j are coordinates in the range [0, 255]. Values outside this range will generate unpredictable results.

*zval* is the floating point value to be written at (i, j) in the Z buffer.

**PMzput** returns a pointer to the next pixel on the given row (i+1,j). This pointer may be used by **PMqzput** for more efficient frame buffer access.

For even faster access the pointer returned can be used directly (unlike the pointer returned from **PMv0put**), since Z buffer memory is fully mapped.

NOTES

Refer to PMzbrk(3X) for page register use.

PMzput does not consider subscreens.

The pointer returned by this function can be cast to allow access to *char*, *int*, or other types of data stored in Z memory.

SEE ALSO

PMputpix(3X) PMqzput(3X) PMv0put(3X)

**printf** – formatted output conversion on host

#### SYNOPSIS

void printf(format [, arg ] ...)
char \*format;

#### DESCRIPTION

**printf** does formatted output in the same way as the UNIX system library **printf**. The full range of flags, widths, precisions, and format specifiers of the UNIX system **printf** is allowed. The actual printing is done by a host program calling **DEVpoll\_nodes**, for example **devprint**, and is displayed on whatever device that process was invoked on.

It is possible to print a floating point number stored in host (IEEE) format with the new modifier, i, that is used in the same way as the l modifier of the UNIX system **printf**. However, because the DSP C compiler generates DSP DA move instructions that can destroy bits when passing an IEEE float to **printf**, the compiler must be fooled into thinking the float is a different data type of the appropriate size that will be passed with a bitwise copy. For example, to print a single host float:

printf("%if", \*(long \*)&host float);

This technique must also be used whenever a float variable that is going to be passed to a function contains data that is not in DSP float format.

A new format specifier %b is also allowed. It formats the argument as an unsigned binary short integer. If %lb is specified, a 32-bit argument is assumed.

**printf** sends a *system message* and its arguments to the host and then returns. Some time later the host processes the format string and reads any pointer data from the nodes via DMA. The node program must therefore be careful not to modify any of the data or page registers associated with pointers in the **printf** argument list. To accomplish this **PMwaitsem** can be called right after **printf** to cause the node to wait until after the host has completely finished its **printf** processing.

#### NOTES

Up to 10 arguments of any scalar type may be given to printf. Using more than 10 arguments causes undefined behavior.

Because ints are 16-bits on the DSP32 and 32-bits on the host, the I modifier must be used when a 32-bit integer quantity is to be printed; for example, to print a float in hex format:

printf("%#IX", f);

#### SEE ALSO

devprint(1) PMwaitsem(3N) DEVpoll\_nodes(3H) printf() on host system

**DEVimage header** - format of a DEVtools image file.

#### SYNOPSIS

#include <devimage.h>

typedef struct

{

l		
unsigned long	magic;	/* magic number to indicate format */
unsigned long	optional header_size;	<pre>/* size of optional header */</pre>
unsigned long	image format;	<pre>/* how the pixels are stored */</pre>
unsigned long	pixel size;	/* number of bytes per pixel */
unsigned long	storage mode;	/* order of pixels in the file */
unsigned long	pixels per line;	/* number of pixels per scan line */
unsigned long	number of lines;	/* number of scan lines */
unsigned long	x offset;	/* initial X value */
unsigned long	y offset;	/* initial Y value */
} DEVimage_header;	<u> </u>	

#### DESCRIPTION

The **DEVimage\_header** structure precedes all data in an image file and specifies information necessary to correctly display the image. **DEVimage\_header** contains only a minimum amount of information about the image. It is assumed that the optional header that follows **DEVimage\_header** will contain more specific information on the file's contents if necessary.

For portability reasons, each member of the structure is stored in the image file as an array of 8 decimal ACSII characters. The two routines **DEVget\_image\_header** and **DEVput\_image\_header** should be used to read/write and convert the image header from/to ASCII. Each of the members of the structure are explained in detail below.

The *magic* member of the structure contains a "magic number" indicating whether this file is in DEVtools image format or not. A value of DEV\_IMAGE\_MAGIC indicates that the file is in DEVtools image format, other values indicate that the format is *not* DEVtools image format.

The *optional\_header\_size* member gives the size of the optional header in bytes. The optional header is placed directly after the image header in the file. If the optional header is not present, this field is 0.

The *image format* field tells how the pixel information is stored in the image file. Valid formats are:

#define DEV_USER_DEFINED	/* user defined image type */
#define DEV_RGBA_PACKED_PIXELS	/* RGBA order, 4 bytes per pixel */
#define DEV_RGB_PACKED_PIXELS	/* RGB order, 3 bytes per pixel */
#define DEV_MONO_PIXELS	/* one byte per pixel */
#define DEV_MONO_R_PIXELS	/* one red byte per pixel */
#define DEV_MONO_G_PIXELS	/* one green byte per pixel */
#define DEV_MONO_B_PIXELS	/* one blue byte per pixel */
#define DEV_MONO_A_PIXELS	/* one alpha byte per pixel */
<pre>#define DEV_MONO_16_PIXELS</pre>	/* 16 bit pixels */
#define DEV_DSP_FLOAT_PIXELS	/* 32 bit DSP floating point pixels */
#define DEV_IEEE_FLOAT_PIXELS	/* 32 bit IEEE floating point pixels */
#define DEV_RGB_PACKED_PIXELS	/* unpacked (16 bit components) RGB pixels */
#define DEV_RGBA_PACKED_PIXELS	/* unpacked (16 bit components) RGBA pixels */
#define DEV_RGB_PACKED_ENCODE	D_PIXELS /* run-length encoded RBG pixels */
#define DEV_ABGR_PACKED_PIXELS	/* packed ABGR pixels */

#define DEV\_RGB\_ENCODED\_PIXELS /\* unpacked, run-length encoded RGB pixels \*/

The *pixel size* field contains the number of bytes that make up a single pixel.

The storage\_mode indicates the order in which the pixels are stored in the image. Valid values for storage\_mode are:

DEV\_ROW\_MAJOR - pixels are stored by rows, that is in the order (0,0), (1,0), (2,0),...,(0,1), (1,1), ...

DEV\_COLUMN\_MAJOR - pixels are stored by columns, that is in the order (0,0), (0,1), (0,2),...,(1,0), (1,1), ...

The pixels per line member indicates the number of pixels per scan line (width) for this image.

The number of lines field indicates how many scan lines (height) are contained in this image.

The x offset field stores the X value of the initial pixel.

The y offset field stores the Y value of the initial pixel.

SEE ALSO

DEVget\_image\_header(3S) DEVput\_image\_header(3S) devsave(1) devdisp(1) picsave(1) picdisp(1) raydisp(1) raysave(1)

PMcommand - data structure used for FIFO commands

SYNOPSIS

#include <pxm.h>

typedef struct {

short opcode; short count; float \*data\_ptr; } PMcmdtype;

extern PMcmdtype PMcommand;

#### DESCRIPTION

Host programs usually operate on the Pixel Machine by sending data packets to the pipe nodes through the FIFOs. The pipe nodes may modify, delete, or pass on the command packets unmodified, or they may also generate new packets. The format of these data packets (called **commands**) is:

OPCODE COUNT PARAM, ... PARAM count

where OPCODE and COUNT are 16-bit values, and each of the parameters in

PARAM .... PARAM count

is a 32-bit value.

The global data structure, PMcommand, defined in both the pipe and pixel nodes, reflects this packet structure. The members of this structure contain the following:

PMcommand.opcode: contains the opcode

PMcommand.count: contains the negated count of the number of bytes in the parameter list

PMcommand.data\_ptr: points to a static buffer containing the parameters. It may be changed to point to a user-defined buffer.

Pipe node programs read a command from the input FIFO in two steps:

call PMgetop to load an opcode and count from the input FIFO

if parameter count is nonzero, call PMgetdata to load parameters from the input FIFO.

Pixel nodes read a command by calling PMgetcmd, which loads all three components of the command.

Pipe node programs may write a command to the output FIFO in two ways. First, by calling **PMputop** followed (if **count** is nonzero) by a call to **PMputdata**. Secondly, by calling **PMputcmd**, which combines the functionality of **PMputop** and **PMputdata**.

By changing members of the PMcommand structure, a pipe node program may modify the command stream as needed.

Pixel node programs read commands from the last pipe node but cannot write commands.

SEE ALSO

DEVwrite(3H) PMgetcmd(3X) PMgetdata(3P)

# DEVtools

# PMgetop(3P) PMputcmd(3P) PMputdata(3P) PMputop(3P)

.

**DEVpipe read** – reads a block of memory from a pipe DSP

# SYNOPSIS

#include <host/devtools.h>

int DEVpipe\_read(pixel\_system, node, addr, buffer, nbytes)
DEVpixel\_system \*pixel\_system;
int node;
DEVushort addr;
DEVbyte \*buffer;
int nbytes;

#### DESCRIPTION

**DEVpipe\_read** reads a block of memory from a pipe DSP. The data is retrieved from DSP memory using parallel DMA.

*pixel\_system* is a pointer to the system descriptor, *node* is the number of the pipe node from which the data is to be read. *addr* is the location in the DSP address space that contains the data is be read. *buffer* points to the location into which the data is to be read. *nbytes* is the number of bytes of data to be read. *nbytes* should always be an even number. If *nbytes* is odd, *nbytes*+1 bytes of data will be read.

No byte order translation is performed. The data read will be in the same byte order as it is in the DSP memory.

**DEVpipe\_read** uses parallel DMA I/O to transfer the data. As a result, the parallel control register is updated by this routine. The parallel communications modes are altered to:

enable DMA

set PAR to be autoincremented on DMA

set the interrupt vector to 16-bit mode

DEVpipe\_read should always return zero.

#### NOTES

If *nbytes* is odd, **DEVpipe\_read** will read *nbytes*+1 bytes of data and return -1 as its return value. The return value should be the number of bytes written, not zero.

DEVieee dsp - convert from the host's floating-point format to the DSP32 floating point format

#### **SYNOPSIS**

#include <host/devtools.h>

DEVulong DEVieee\_dsp(f) double f;

# DESCRIPTION

The host and the DSP32 use different formats for floating point numbers. **DEVieee\_dsp** converts a single floating point number in the IEEE format used by the host to a 32 bit floating point number in DSP32 format. The number to be converted is stored in f.

The value returned by **DEVieee\_dsp** must be converted to the correct Pixel Machine byte order. This is done implicitly when the value is written to the pipe, but it must de done explicitly using **DEVbswapl()** or **DEVswap long()** if the value is sent to the Pixel Machine in some other way (e.g., via DMA).

## RETURNS

DEVieee\_dsp returns a 32 bit number in the DSP32 floating point format.

### NOTES

DSP floating point values should always be treated as unsigned long values on the host to prevent the compiler from performing undesired type-casting; for example, promotion to double when used as a function argument.

#### SEE ALSO

DEVdsp\_ieee(3S) DEVbswapl(3S) DEVswap\_long(3S) DEVswap\_short(3S)