

# **SciTech Software Inc.**

180 E 4<sup>th</sup> St, Suite 300 • Chico, CA 95928 • ☎ 530-894-8400 ■ 530-894-9069  
www.scitechsoft.com • ftp.scitechsoft.com • sales@scitechsoft.com

## **SciTech SNAP Graphics Architecture**

**Version: 2.3**

**Revision Date:** Tuesday, January 20, 2004

**by Kendall Bennett**

This document defines the interface for an operating system portable, 32-bit loadable device driver architecture that will provide access to accelerated graphics hardware. The specification itself targets any personal computer environment regardless of the processor type, however a particular binary driver is only compatible with the processor it was compiled for. Some of the accelerator functions supported include hardware cursors, multi buffering, solid and transparent off-screen bitmaps, rectangle filling and line drawing.

### ***Intellectual Property***

Copyright © 1997-2004 SciTech Software, Inc - All Rights Reserved. Duplication and distribution of this document is strictly prohibited without prior, written consent from SciTech Software, Inc.

While every precaution has been taken in the preparation of this document, SciTech Software assume no responsibility for errors or omissions, and make no warranties, expressed or implied, of functionality or suitability for any purpose.

### ***Trademarks***

All trademarks used in this document are property of their respective owners.

# Table of Contents

---

<b>Introduction .....</b>	<b>2</b>
What is SciTech SNAP? .....	2
What is SciTech SNAP Graphics Architecture? .....	2
<b>Installing SciTech SNAP Graphics .....</b>	<b>4</b>
Downloading and Installing SciTech SNAP Graphics .....	4
Makefile Utilities Configuration .....	5
DOS/Windows hosted tools (start-sdk.bat) .....	5
Win32 hosted tools (start-sdk.bat) .....	7
Windows hosted tools for RTTarget-32 (start-sdk.bat) .....	7
OS/2 hosted tools (start-sdk.cmd) .....	8
Linux hosted tools (start-sdk.linux) .....	9
QNX hosted tools (start-sdk.qnx) .....	9
Compiling SciTech SNAP Graphics .....	10
Compiling release and debug builds .....	10
Compiling the sample programs .....	11
Setting up Your Compiler Configuration .....	11
Using the Makefile Utilities .....	13
Standard Makefile Targets .....	13
Standard Makefile Options .....	14
CauseWay DOS Extender Support .....	15
Connecting with Perforce .....	15
Download a Perforce Client .....	15
Setting up your environment for anonymous access .....	16
Setting up your client mapping .....	16
Syncing up for the first time .....	17
Using Perforce from the command line .....	18
<b>Programming with SNAP Graphics .....</b>	<b>19</b>
Loading and Initializing SciTech SNAP Graphics .....	19
Runtime Library Standard Locations .....	19
Enumerating Installed Devices and Loading a Driver .....	20
Locating and Calling Device Driver Functions .....	20
Querying Device Configuration Information .....	21
Working With Display Modes .....	21
Finding Available Display Modes .....	21
Refresh Rate Control .....	22
Using Custom Display Modes .....	23
2D Coordinate System .....	23
Multi Buffering .....	25
Accessing Offscreen Video Memory .....	26
Virtual Buffer Scrolling .....	28
Palette Programming During Double Buffering .....	28
Integer Coordinates .....	28
Color Values .....	28
Direct Framebuffer Access .....	29
Hardware Triple Buffering .....	29
Using the Buffer Manager .....	30

Hardware Video Overlay Functions .....	31
Stereoscopic Liquid Crystal Shutter Glasses .....	32
Refresh rates and stereoscopic imaging .....	33
Software driven display start address swapping .....	33
Developing for Maximum Compatibility .....	33
Support Both 15-bit and 16-bits Per Pixel Modes.....	34
Support Both 24-bit and 32-bits per Pixel Modes.....	34
Do Not Assume Support for Double Scanned Modes .....	34
<b>Graphics Device Driver Overview.....</b>	<b>35</b>
Overview of Global Functions.....	35
Driver Loading and Initialization Functions .....	35
Display Mode Management Functions .....	35
Rectangle Arithmetic Functions.....	36
Monitor Detection Functions.....	36
Monitor Database Functions.....	36
Monitor Command Set Functions.....	36
Overview of Queried Function Groups .....	37
Display Driver Initialization Functions .....	37
Device Driver Control Functions.....	38
2D Rendering State Functions.....	38
2D Drawing Functions .....	38
Buffer Manager Functions .....	40
Complex Region Management Functions .....	40
Hardware Video Overlay Functions .....	40
Hardware Cursor Functions .....	41
<b>Graphics Device Driver Reference .....</b>	<b>43</b>
External Functions.....	44
DDC_init .....	45
DDC_initExt.....	46
DDC_readEDID .....	47
DDC_writeEDID .....	48
EDID_parse.....	49
GA_addMode .....	50
GA_addRefresh.....	51
GA_computeCRTCTimings.....	52
GA_delMode.....	53
GA_detectPnPMonitor .....	54
GA_disableVBEMode.....	55
GA_disjointRect .....	56
GA_emptyRect .....	57
GA_enableVBEMode .....	58
GA_enumerateDevices .....	59
GA_equalRect.....	60
GA_errorMsg .....	61
GA_getCRTCTimings.....	62
GA_getCurrentRef2d .....	63
GA_getDaysLeft.....	64
GA_getDisplaySerialNo.....	65
GA_getDisplayUserName.....	66
GA_getFakePCIID .....	67
GA_getGlobalOptions .....	68
GA_getInternalName.....	69

GA_getLicensedDevices .....	70
GA_getMaxRefreshRate.....	71
GA_getParsedEDID.....	72
GA_getSNAPConfigPath.....	73
GA_insetRect .....	74
GA_isLiteVersion .....	75
GA_isOEMVersion .....	76
GA_isSharedDriverLoaded.....	77
GA_isSimpleRegion .....	78
GA_loadDriver.....	79
GA_loadInGUI.....	80
GA_loadModeProfile .....	81
GA_loadRef2d .....	82
GA_loadRegionMgr .....	83
GA_offsetRect.....	84
GA_programMTRRegisters .....	85
GA_ptInRect .....	86
GA_queryFunctions .....	87
GA_readGlobalOptions .....	89
GA_registerLicense .....	90
GA_restoreCRTCTimings.....	91
GA_saveCRTCTimings.....	92
GA_saveGlobalOptions .....	93
GA_saveModeProfile.....	94
GA_saveMonitorInfo.....	95
GA_saveOptions .....	96
GA_sectRect.....	97
GA_sectRectCoord .....	98
GA_sectRectFast .....	99
GA_sectRectFastCoord.....	100
GA_setActiveDevice.....	101
GA_setCRTCTimings .....	102
GA_setDefaultRefresh.....	103
GA_setGlobalOptions .....	104
GA_setMinimumDriverVersion .....	105
GA_softStereoExit.....	106
GA_softStereoGetFlipStatus .....	107
GA_softStereoInit.....	108
GA_softStereoOff .....	109
GA_softStereoOn .....	110
GA_softStereoScheduleFlip .....	111
GA_softStereoWaitTillFlipped.....	112
GA_status.....	113
GA_unionRect.....	114
GA_unionRectCoord .....	115
GA_unloadDriver .....	116
GA_unloadRef2d .....	117
GA_unloadRegionMgr.....	118
GA_useDoubleScan.....	119
MCS_begin .....	120
MCS_beginExt .....	121
MCS_enableControl .....	122
MCS_end.....	123

MCS_getCapabilitiesString .....	124
MCS_getControlMax.....	125
MCS_getControlValue.....	126
MCS_getControlValues .....	127
MCS_getSelfTestReport.....	128
MCS_getTimingReport.....	129
MCS_isControlSupported.....	130
MCS_resetControl.....	131
MCS_saveCurrentSettings .....	132
MCS_setControlValue .....	133
MCS_setControlValues.....	134
MDBX_close .....	135
MDBX_first.....	136
MDBX_flush.....	137
MDBX_getErrCode.....	138
MDBX_getErrorMsg.....	139
MDBX_importINF .....	140
MDBX_insert.....	141
MDBX_last .....	142
MDBX_next.....	143
MDBX_open .....	144
MDBX_prev.....	145
MDBX_update .....	146
PE_freeLibrary .....	147
PE_getError .....	148
PE_getFileSize.....	149
PE_getProcAddress.....	150
PE_loadLibrary.....	151
PE_loadLibraryExt.....	152
PE_loadLibraryMGL.....	153
REF2D_loadDriver .....	154
REF2D_queryFunctions .....	155
REF2D_unloadDriver.....	156
Type Definitions .....	157
DDC_ChannelsType .....	158
DDC_DPMSStatesType.....	159
DDC_SCIFlagsType .....	160
DDC_errCode.....	161
EDID_detailedTiming.....	162
EDID_displayTypes.....	163
EDID_flags.....	164
EDID_maxResCodes .....	165
EDID_record .....	166
EDID_signalLevels .....	168
EDID_standardTiming .....	169
EDID_timingTypes.....	170
GA_2DRenderFuncs.....	171
BitBlt.....	172
BitBltBM.....	173
BitBltColorPatt .....	175
BitBltColorPattBM .....	176
BitBltColorPattLin.....	177
BitBltColorPattSys.....	178

BitBltEx.....	179
BitBltExBM.....	181
BitBltExLin.....	183
BitBltExSys.....	185
BitBltExTest.....	187
BitBltLin.....	188
BitBltPatt.....	190
BitBltPattBM.....	191
BitBltPattLin.....	192
BitBltPattSys.....	193
BitBltPlaneMasked.....	194
BitBltPlaneMaskedBM.....	195
BitBltPlaneMaskedLin.....	197
BitBltPlaneMaskedSys.....	199
BitBltSys.....	201
ClipEllipse.....	202
ClipMonoImageLSBBM.....	203
ClipMonoImageLSBLin.....	204
ClipMonoImageLSBSys.....	205
ClipMonoImageMSBBM.....	206
ClipMonoImageMSBLin.....	207
ClipMonoImageMSBSys.....	208
DrawBresenhamLine.....	209
DrawBresenhamStippleLine.....	211
DrawBresenhamStyleLine.....	213
DrawClippedBresenhamLine.....	215
DrawClippedBresenhamStippleLine.....	217
DrawClippedBresenhamStyleLine.....	219
DrawClippedLineInt.....	221
DrawClippedStippleLineInt.....	222
DrawClippedStyleLineInt.....	223
DrawColorPattEllipseList.....	224
DrawColorPattFatEllipseList.....	225
DrawColorPattRect.....	227
DrawColorPattScanList.....	228
DrawColorPattTrap.....	229
DrawEllipse.....	230
DrawEllipseList.....	231
DrawFatEllipseList.....	232
DrawLineInt.....	234
DrawPattEllipseList.....	235
DrawPattFatEllipseList.....	236
DrawPattRect.....	238
DrawPattScanList.....	239
DrawPattTrap.....	240
DrawRect.....	241
DrawRectExt.....	242
DrawRectLin.....	243
DrawScanList.....	244
DrawStippleLineInt.....	245
DrawStyleLineInt.....	246
DrawTrap.....	247
DstTransBlt.....	248

<i>DstTransBltBM</i> .....	249
<i>DstTransBltLin</i> .....	251
<i>DstTransBltSys</i> .....	253
<i>GetBitmapBM</i> .....	255
<i>GetBitmapSys</i> .....	256
<i>GetPixel</i> .....	257
<i>PutMonoImageLSBBM</i> .....	258
<i>PutMonoImageLSBLin</i> .....	259
<i>PutMonoImageLSBSys</i> .....	260
<i>PutMonoImageMSBBM</i> .....	261
<i>PutMonoImageMSBLin</i> .....	262
<i>PutMonoImageMSBSys</i> .....	263
<i>PutPixel</i> .....	265
<i>SrcTransBlt</i> .....	266
<i>SrcTransBltBM</i> .....	267
<i>SrcTransBltLin</i> .....	269
<i>SrcTransBltSys</i> .....	271
<i>StretchBlt</i> .....	273
<i>StretchBltBM</i> .....	275
<i>StretchBltLin</i> .....	277
<i>StretchBltSys</i> .....	279
<i>UpdateScreen</i> .....	281
<i>GA_2DStateFuncs</i> .....	282
<i>BuildTranslateVector</i> .....	283
<i>DisableDirectAccess</i> .....	284
<i>EnableDirectAccess</i> .....	285
<i>IsIdle</i> .....	286
<i>Set8x8ColorPattern</i> .....	287
<i>Set8x8MonoPattern</i> .....	288
<i>SetAlphaValue</i> .....	289
<i>SetBackColor</i> .....	290
<i>SetBlendFunc</i> .....	291
<i>SetDrawBuffer</i> .....	292
<i>SetForeColor</i> .....	293
<i>SetLineStipple</i> .....	294
<i>SetLineStippleCount</i> .....	295
<i>SetLineStyle</i> .....	296
<i>SetMix</i> .....	297
<i>SetPlaneMask</i> .....	298
<i>Use8x8ColorPattern</i> .....	299
<i>Use8x8MonoPattern</i> .....	300
<i>Use8x8TransColorPattern</i> .....	301
<i>Use8x8TransMonoPattern</i> .....	302
<i>WaitTillIdle</i> .....	303
<i>GA_AccelFlagsType</i> .....	304
<i>GA_AttributeExtFlagsType</i> .....	305
<i>GA_AttributeFlagsType</i> .....	306
<i>GA_BitBltFxFxFlagsType</i> .....	310
<i>GA_BresenhamLineFlagsType</i> .....	313
<i>GA_BufferFlagsType</i> .....	314
<i>GA_CRTInfo</i> .....	317
<i>GA_CRTInfoFlagsType</i> .....	319
<i>GA_CertifyFlagsType</i> .....	320



GA_DPMSFuncs .....	322
DPMSdetect .....	323
DPMSsetState .....	324
GA_LCDUseBIOSFlagsType .....	325
GA_MakeVisibleBufferFlagsType .....	326
GA_OutputFlagsType .....	327
GA_SCIFuncs .....	328
SCIbegin .....	329
SCIdetect .....	330
SCIend .....	331
SCIreadSCL .....	332
SCIreadSDA .....	333
SCIwriteSCL .....	334
SCIwriteSDA .....	335
GA_TVParams .....	336
GA_VBEFuncs .....	337
GetPaletteData .....	338
Set8BitDAC .....	339
SetBytesPerLine .....	340
SetPaletteData .....	341
GA_VideoBufferFormatsType .....	342
GA_VideoOutputFlagsType .....	345
GA_WorkAroundsFlagsType .....	347
GA_blendFuncType .....	350
GA_bltFx .....	353
GA_buf .....	355
GA_buffer .....	357
GA_bufferFuncs .....	358
AllocBuffer .....	359
BitBltBuf .....	360
BitBltColorPattBuf .....	361
BitBltExBuf .....	362
BitBltPattBuf .....	364
BitBltPlaneMaskedBuf .....	365
DrawRectBuf .....	366
DstTransBltBuf .....	367
FlipToBuffer .....	368
FlipToStereoBuffer .....	369
FreeBuffer .....	370
GetClipper .....	371
GetFlipStatus .....	372
GetFlippableBuffer .....	373
GetPrimaryBuffer .....	374
InitBuffers .....	375
LockBuffer .....	376
SetActiveBuffer .....	377
SetClipper .....	378
SrcTransBltBuf .....	379
StretchBltBuf .....	380
UnlockBuffer .....	382
UpdateCache .....	383
UpdateFromCache .....	384
WaitTillFlipped .....	385

GA_busType.....	386
GA_certifyChipInfo.....	387
GA_certifyInfo.....	388
GA_clipper.....	389
GA_clipperFuncs.....	390
CreateClipper.....	391
DestroyClipper.....	392
GetClipList.....	393
IsClipListChanged.....	394
GA_color.....	395
GA_colorCursor.....	396
GA_colorCursor256.....	397
GA_colorCursorRGB.....	398
GA_colorCursorRGBA.....	399
GA_colorPattern.....	400
GA_colorPattern_1.....	401
GA_colorPattern_16.....	402
GA_colorPattern_24.....	403
GA_colorPattern_32.....	404
GA_colorPattern_4.....	405
GA_colorPattern_8.....	406
GA_configInfo.....	407
GA_cursorFuncs.....	408
BeginAccess.....	409
EndAccess.....	410
IsHardwareCursor.....	411
SetColorCursor.....	412
SetColorCursor256.....	413
SetColorCursorRGB.....	414
SetColorCursorRGBA.....	415
SetCursorPos.....	416
SetMonoCursor.....	417
SetMonoCursorColor.....	418
ShowCursor.....	419
GA_devCtx.....	420
GA_driverFuncs.....	425
EnableStereoMode.....	426
GetCurrentScanLine.....	427
GetDisplayStartStatus.....	428
GetGammaCorrectData.....	429
GetGammaCorrectDataExt.....	430
GetPaletteData.....	431
GetPaletteDataExt.....	432
GetVSyncWidth.....	433
IsVSync.....	434
SetBank.....	435
SetDisplayStart.....	436
SetDisplayStartXY.....	438
SetGammaCorrectData.....	439
SetGammaCorrectDataExt.....	440
SetPaletteData.....	441
SetPaletteDataExt.....	442
SetStereoDisplayStart.....	443

SetVSyncWidth.....	444
WaitVSync.....	445
GA_funcGroupsType .....	446
GA_globalOptions.....	448
GA_initFuncs.....	450
AlignLinearBuffer .....	451
GetActiveHead .....	452
GetCRTCTimings .....	453
GetCertifyInfo .....	454
GetClosestPixelClock .....	455
GetConfigInfo.....	457
GetCurrentRefreshRate.....	458
GetCurrentVideoModeInfo .....	459
GetCustomVideoModeInfo.....	460
GetCustomVideoModeInfoExt .....	461
GetDisplayOutput .....	462
GetMonitorInfo .....	463
GetNumberOfHeads.....	464
GetOptions .....	465
GetUniqueFilename .....	466
GetVideoMode.....	467
GetVideoModeInfo .....	468
GetVideoModeInfoExt.....	469
PerformDisplaySwitch .....	470
PollForDisplaySwitch .....	471
SaveCRTCTimings .....	472
SaveRestoreState .....	473
SetActiveHead.....	474
SetCRTCTimings.....	475
SetCustomVideoMode.....	476
SetDisplayOutput.....	478
SetGlobalRefresh .....	479
SetModeProfile .....	480
SetMonitorInfo.....	481
SetOptions.....	482
SetRef2dPointer .....	483
SetSoftwareRenderFuncs .....	484
SetVideoMode .....	485
SwitchPhysicalResolution.....	488
GA_largeInteger.....	489
GA_layout .....	490
GA_loaderFuncs.....	491
InitDriver.....	492
QueryFunctions .....	493
UnloadDriver.....	495
GA_mixCodesType.....	496
GA_mode.....	498
GA_modeFlagsType .....	499
GA_modeInfo .....	500
GA_modeProfile .....	506
GA_monitor .....	507
GA_monitorFlagsType.....	508
GA_monoCursor .....	509

GA_multiHeadType .....	510
GA_options .....	511
GA_palette .....	517
GA_paletteExt .....	518
GA_pattern .....	519
GA_pixelFormat .....	520
GA_recMode .....	523
GA_rect .....	524
GA_region .....	525
GA_regionFuncs .....	526
ClearRegion .....	527
CopyIntoRegion .....	528
CopyRegion .....	529
DiffRegion .....	530
DiffRegionRect .....	531
FreeRegion .....	532
IsEmptyRegion .....	533
IsEqualRegion .....	534
NewRectRegion .....	535
NewRegion .....	536
OffsetRegion .....	537
OptimizeRegion .....	538
PtInRegion .....	539
SectRegion .....	540
SectRegionRect .....	541
TraverseRegion .....	542
UnionRegion .....	543
UnionRegionOfs .....	544
UnionRegionRect .....	545
GA_rop3CodesType .....	546
GA_segment .....	551
GA_span .....	552
GA_stipple .....	553
GA_trap .....	554
GA_videoFuncs .....	555
AllocVideoBuffer .....	556
EndVideoFrame .....	557
FreeVideoBuffer .....	558
SetVideoColorKey .....	559
SetVideoOutput .....	560
StartVideoFrame .....	561
GA_videoInf .....	562
MCS_controlsType .....	563
MCS_polarityFlagsType .....	565
MDBX_errCodes .....	566
N_errorType .....	567
N_fix32 .....	568
N_float32 .....	569
N_int16 .....	570
N_int32 .....	571
N_int8 .....	572
N_physAddr .....	573
N_uint16 .....	574

N_uint32 .....	575
N_uint8 .....	576
PE_errorCodes .....	577
REF2D_driver .....	578
DrawRectExtSW .....	579
ForceSoftwareOnly .....	580
PostSwitchPhysicalResolution .....	581
QueryFunctions .....	582
RotateBitmap .....	583
SetColorCompareMask .....	584
SetDrawBuffer .....	585
SetDrawSurface .....	586
<b>PM Library Reference .....</b>	<b>587</b>
External Functions .....	588
CPU_getProcessorName .....	589
CPU_getProcessorSpeed .....	590
CPU_getProcessorSpeedInHZ .....	591
CPU_getProcessorType .....	592
CPU_have3DNow .....	593
CPU_haveMMX .....	594
CPU_haveRDTSC .....	595
CPU_haveSSE .....	596
EVT_allowLEDS .....	597
EVT_asciiCode .....	598
EVT_flush .....	599
EVT_getCodePage .....	600
EVT_getHeartBeatCallback .....	601
EVT_getMousePos .....	602
EVT_getNext .....	603
EVT_halt .....	604
EVT_isKeyDown .....	605
EVT_joyIsPresent .....	606
EVT_joySetCenter .....	607
EVT_joySetLowerRight .....	608
EVT_joySetUpperLeft .....	609
EVT_peekNext .....	610
EVT_pollJoystick .....	611
EVT_post .....	612
EVT_repeatCount .....	613
EVT_scanCode .....	614
EVT_setCodePage .....	615
EVT_setHeartBeatCallback .....	616
EVT_setMousePos .....	617
EVT_setUserEventFilter .....	618
LZTimerCount .....	619
LZTimerCountExt .....	620
LZTimerLap .....	621
LZTimerLapExt .....	622
LZTimerOff .....	623
LZTimerOffExt .....	624
LZTimerOn .....	625
LZTimerOnExt .....	626
PCI_accessReg .....	627

<i>PCI_enumerate</i> .....	628
<i>PCI_getNumDevices</i> .....	629
<i>PCI_readRegBlock</i> .....	630
<i>PCI_writeRegBlock</i> .....	631
<i>PE_freeLibrary</i> .....	632
<i>PE_getError</i> .....	633
<i>PE_getFileSize</i> .....	634
<i>PE_getProcAddress</i> .....	635
<i>PE_loadLibrary</i> .....	636
<i>PE_loadLibraryExt</i> .....	637
<i>PE_loadLibraryMGL</i> .....	638
<i>PM_agpCommitPhysical</i> .....	639
<i>PM_agpExit</i> .....	640
<i>PM_agpFreePhysical</i> .....	641
<i>PM_agpInit</i> .....	642
<i>PM_agpReleasePhysical</i> .....	643
<i>PM_agpReservePhysical</i> .....	644
<i>PM_allocLockedMem</i> .....	645
<i>PM_allocPage</i> .....	646
<i>PM_allocRealSeg</i> .....	647
<i>PM_backslash</i> .....	648
<i>PM_blockUntilTimeout</i> .....	649
<i>PM_callRealMode</i> .....	650
<i>PM_calloc</i> .....	651
<i>PM_closeConsole</i> .....	652
<i>PM_doSuspendApp</i> .....	653
<i>PM_enableWriteCombine</i> .....	654
<i>PM_enumWriteCombine</i> .....	655
<i>PM_fatalError</i> .....	656
<i>PM_findBPD</i> .....	657
<i>PM_findClose</i> .....	658
<i>PM_findFirstFile</i> .....	659
<i>PM_findNextFile</i> .....	660
<i>PM_flushTLB</i> .....	661
<i>PM_free</i> .....	662
<i>PM_freeLibrary</i> .....	663
<i>PM_freeLockedMem</i> .....	664
<i>PM_freePage</i> .....	665
<i>PM_freePhysicalAddr</i> .....	666
<i>PM_freeRealSeg</i> .....	667
<i>PM_freeShared</i> .....	668
<i>PM_getA0000Pointer</i> .....	669
<i>PM_getBIOSPointer</i> .....	670
<i>PM_getBootDrive</i> .....	671
<i>PM_getCOMPort</i> .....	672
<i>PM_getConsoleStateSize</i> .....	673
<i>PM_getCurrentPath</i> .....	674
<i>PM_getDirectDrawWindow</i> .....	675
<i>PM_getFileAttr</i> .....	676
<i>PM_getFileTime</i> .....	677
<i>PM_getIOPL</i> .....	678
<i>PM_getLPTPort</i> .....	679
<i>PM_getMachineName</i> .....	680

PM_getOSName .....	681
PM_getOSType.....	682
PM_getPhysicalAddr .....	683
PM_getPhysicalAddrRange .....	684
PM_getProcAddress.....	685
PM_getSNAPConfigPath .....	686
PM_getSNAPPath.....	687
PM_getUniqueID.....	688
PM_getVESABuf.....	689
PM_getVGASize.....	690
PM_getch .....	691
PM_getdcwd .....	692
PM_haveBIOSAccess.....	693
PM_init .....	694
PM_inpb.....	695
PM_inpd.....	696
PM_inpw.....	697
PM_installService .....	698
PM_installServiceExt .....	699
PM_int86 .....	700
PM_int86x .....	701
PM_isSDDActive .....	702
PM_kbhit.....	703
PM_loadDirectDraw.....	704
PM_loadLibrary .....	705
PM_lockCodePages .....	706
PM_lockDataPages .....	707
PM_makepath.....	708
PM_malloc .....	709
PM_mallocShared .....	710
PM_mapPhysicalAddr .....	711
PM_mapRealPointer .....	712
PM_mkdir .....	713
PM_openConsole.....	714
PM_outpb.....	715
PM_outpd.....	716
PM_outpw.....	717
PM_realloc .....	718
PM_removeService.....	719
PM_restartRealTimeClock .....	720
PM_restoreConsoleState .....	721
PM_restoreRealTimeClockHandler.....	722
PM_restoreThreadPriority .....	723
PM_restoreVGASize .....	724
PM_rmdir.....	725
PM_runningInAWindow .....	726
PM_saveConsoleState .....	727
PM_saveVGASize .....	728
PM_setDebugLog.....	729
PM_setFatalErrorCleanup .....	730
PM_setFileAttr .....	731
PM_setFileTime .....	732
PM_setIOPL .....	733

PM_setLocalBPDPath.....	734
PM_setMaxThreadPriority.....	735
PM_setOSCursorLocation.....	736
PM_setOSScreenWidth.....	737
PM_setRealTimeClockFrequency.....	738
PM_setRealTimeClockHandler.....	739
PM_setSuspendAppCallback.....	740
PM_sleep.....	741
PM_splitpath.....	742
PM_startService.....	743
PM_stopRealTimeClock.....	744
PM_stopService.....	745
PM_unloadDirectDraw.....	746
PM_unlockCodePages.....	747
PM_unlockDataPages.....	748
PM_useLocalMalloc.....	749
ULZElapsedTime.....	750
ULZReadTime.....	751
ULZTimerCount.....	752
ULZTimerLap.....	753
ULZTimerOff.....	754
ULZTimerOn.....	755
ULZTimerResolution.....	756
ZTimerInit.....	757
ZTimerInitExt.....	758
Type Definitions.....	759
CPU_largeInteger.....	760
CPU_processorType.....	761
EVT_asciiCodesType.....	763
EVT_eventJoyAxisType.....	766
EVT_eventJoyMaskType.....	767
EVT_eventMaskType.....	768
EVT_eventModMaskType.....	769
EVT_eventMouseMaskType.....	770
EVT_eventType.....	771
EVT_masksType.....	772
EVT_scanCodesType.....	773
LZTimerObject.....	776
PCIAGPCapability.....	777
PCIAGPCommand.....	778
PCIAGPStatus.....	779
PCIAccessRegFlags.....	780
PCICapsHeader.....	781
PCICapsType.....	782
PCIClassTypes.....	783
PCICommandFlags.....	784
PCIDeviceInfo.....	785
PCIHeaderTypeFlags.....	787
PCIStatusFlags.....	788
PCIType0Info.....	789
PCIType1Info.....	791
PCIType2Info.....	793
PCIslot.....	794



<i>PE_errorCodes</i> .....	795
<i>PMBYTE_REGS</i> .....	796
<i>PMDWORD_REGS</i> .....	797
<i>PMEnableWriteCombineErrors</i> .....	798
<i>PMEnableWriteCombineFlags</i> .....	799
<i>PMFileFlagsType</i> .....	800
<i>PM_REGS</i> .....	801
<i>PMS_REGS</i> .....	802
<i>PMSplitPathFlags</i> .....	803
<i>PMWORD_REGS</i> .....	804
<i>PM_HWND</i> .....	805
<i>PM_IRQHandle</i> .....	806
<i>PM_MODULE</i> .....	807
<i>PM_agpMemoryType</i> .....	808
<i>PM_enumWriteCombine_t</i> .....	809
<i>PM_fatalCleanupHandler</i> .....	810
<i>PM_findData</i> .....	811
<i>PM_intHandler</i> .....	812
<i>PM_irqHandler</i> .....	813
<i>PM_lockHandle</i> .....	814
<i>PM_physAddr</i> .....	815
<i>PM_suspendAppCodesType</i> .....	816
<i>PM_suspendAppFlagsType</i> .....	817
<i>PM_suspendApp_cb</i> .....	818
<i>PM_time</i> .....	819
<i>RM_REGS</i> .....	820
<i>RMS_REGS</i> .....	821
<i>__codePtr</i> .....	822
<i>codepage_entry_t</i> .....	823
<i>codepage_t</i> .....	824
<i>event_t</i> .....	825
<b>Index</b> .....	<b>827</b>



---

# Introduction

---

This document contains the SciTech SNAP Graphics Architecture Hardware Abstraction Layer reference.

---

## What is SciTech SNAP?

---

The SciTech System Neutral Access Protocol, or SNAP, is an operating system portable, dynamically loadable, 32-bit device driver architecture. SciTech SNAP defines the architecture for loading an operating system neutral binary device driver for any type of hardware device, be it a graphics controller, audio controller, SCSI controller or network controller. SciTech SNAP drivers are source code portable between different microprocessor platforms, and the binary drivers are operating system portable within a particular microprocessor family. Hence the Intel x86 drivers can work on any 386+ CPU with any 32-bit operating system or environment supported on that CPU.

The main SciTech SNAP library for a particular device type is always contained in a single file that we call a 'Binary Portable DLL', or .bpd file. In the case of the Graphics Architecture, the driver file is named 'graphics.bpd' and always lives in the operating system specific SciTech SNAP driver directory, or in the drivers directory of the application using it. Internally the complete set of device support files may be spread across multiple files inside subdirectories, or all files may be optionally bound into a single, large binary file. The internal architecture of SNAP is such that only the necessary code to communicate with a specific device is actually loaded into memory at runtime, regardless of whether everything is bound into a single large binary file or maintained separately on disk. Loading only the necessary portions at runtime keeps the runtime memory footprint for the drivers exceptionally low. For instance even though the complete, statically bound graphics driver file may be well in excess of 8Mb, only a small portion of that (say around 250Kb) is actually loaded in memory at runtime to support a particular graphics device.

Due to the dynamic nature of the SciTech SNAP architecture, it is possible to bind together support for only a few specific hardware devices in a single binary library. For instance an industrial developer may only require support for the three graphics chipsets they install in their machines on a production line, so the total static size of the SNAP binaries can be significantly smaller. This can help save on disk and flash ROM space in highly embedded and industrial applications. SciTech SNAP also allows certain features to be 'bound out' of the resulting binary, further reducing disk space if those features are not required for a particular environment.

---

## What is SciTech SNAP Graphics Architecture?

---

This particular specification deals with graphics controllers and is called the SciTech SNAP Graphics Architecture. This document defines a complete Hardware Abstraction

Layer (HAL) for modern graphics controllers, including support for features such as 2D and 3D rendering, video acceleration, power management and Plug and Play operation. The following is a brief list of some of the features provided by this specification:

- Standard application interface to graphical user interface accelerator devices.
- Operating system neutral high performance 32-bit loadable device driver.
- Support for Plug and Play and multiple independent controllers in a single system.
- Support for bus mastering (i.e.: PCI and AGP bus interfaces)
- Support for hardware video acceleration.
- Support for 2D hardware acceleration.
- Support for 3D hardware acceleration.
- Support for standard and extended text modes.
- Support for pixel depths from 4 bit to 32-bits per pixel.
- Support for all 16 ROP2 mixes.
- Support for all 256 ROP3 mixes.
- Support for off screen memory management for bitmap storage.
- Support for multi buffering for flicker free animation.
- Support for virtual scrolling for large desktops and arcade style games.
- Support for refresh rate control.
- Support for stereo liquid crystal shutter glasses.
- Support for DPMS Power Management
- Support for I<sup>2</sup>C Serial Control Interface

# Installing SciTech SNAP Graphics

---

This chapter is intended to provide the necessary information to help developers get up and running with the SciTech SNAP Graphics SDK, and to show the steps necessary to compile and link the libraries and sample programs from for your OS and compiler configuration.

## Downloading and Installing SciTech SNAP Graphics

---

Before you install any SciTech Software developer products, you should decide upon a standard root directory for installing all of the products into. By default the many of the configuration files assume `c:\scitech` as the installation location (`~/scitech` for Unix). You might like to install the files onto a different drive or directory, but should install all the files for all the different SciTech Software products that you have under the same directory tree. Many SciTech Software products use common libraries and common header files, so when you install them into the same directory you will only have one copy of each of these common files and won't run into conflicts with multiple copies of the same files on your system.

Once you have decided on a location to install to, at a minimum you will need a copy of the source archive, the base utilities archive for your operating system and the SciTech SNAP device driver binaries. Uncompress all files in the archive into the directory on your system where you want the files to live (normally `c:\scitech` or `~/scitech` for Unix). You can optionally install the source code for the base archive utilities as well as the SDK documentation in HTML format (for offline browsing). The names of the archives included in this release are:

DOS, Windows and OS/2 hosted files	
<code>snap_sdk_2.1-r13-src.zip</code>	Source archive in DOS/Win32 format
<code>snap_sdk_2.1-r13-dos.zip</code>	DOS hosted base utilities
<code>snap_sdk_2.1-r13-win32.zip</code>	Win32 hosted base utilities
<code>snap_sdk_2.1-r13-os2.zip</code>	OS/2 hosted base utilities
<code>snap_sdk_2.1-r13-drivers.zip</code>	SciTech SNAP device driver binaries
<code>snap_sdk_2.1-r13-util.zip</code>	Source code for base utilities
<code>snap_sdk_2.1-r13-docs.zip</code>	SDK documentation in HTML format

Linux, QNX and Unix hosted files	
snap_sdk_2.1-r13-src.tar.gz	Source archive in Unix format
snap_sdk_2.1-r13-linux.tar.gz	Linux hosted base utilities
snap_sdk_2.1-r13-qnx.tar.gz	QNX hosted base utilities
snap_sdk_2.1-r13-drivers.tar.gz	SciTech SNAP device driver binaries
snap_sdk_2.1-r13-util.tar.gz	Source code for base utilities
snap_sdk_2.1-r13-docs.tar.gz	SDK documentation in HTML format

## Makefile Utilities Configuration

---

Once you have installed the files, you need to configure the makefile utilities and tools to allow you to compile the libraries. If you are compiling under DOS or a Windows DOS box, you will need to set the environment for your DOS box to at least 4096 bytes as we use a lot of environment variables.

The first step to set up the compile environment is always to edit the set-vars script files specific to your platform (copy it to a different named file if you are working with source code checked out of Perforce as this file will be read only). This file contains two very important variables that you need to ensure are set correctly. These are:

SCITECH	Points to the location where you installed the files
SCITECH_LIB	Should be the same as the above

The SCITECH\_LIB variable is where files get copied to when you do a `dmake install`, and is usually the same as the SCITECH variable. However it can be different if you wish the libraries to be installed to a different path such as on a shared file server for release builds.

To help you get started configuring your system we have created a number of startup script files for each of the different operating systems. These files are called in the `%SCITECH%\start-sdk.*` files with an extension specific to each OS. These files contain the necessary sequence of steps to call the appropriate batch files to set up the compile environment. To use this copy this file to a new filename and edit to call your version of the `set-vars` script file as necessary.

### DOS/Windows hosted tools (start-sdk.bat)

The first thing you need to do is edit the `bin\set-vars.bat` batch file to reference the location where you have installed the files, and the locations where all your compilers are installed. See the comments in `set-vars.bat` for more information.

Once you have the startup file configured, you then need to run the following each time you start a command shell to enable the SciTech makefile utilities (a good idea to put into your startup batch files):

```
call c:\scitech\bin\set-vars.bat
call c:\scitech\bin\wc11-w32.bat
```

The second batch file sets up the compiler configuration for your default compiler. The line above sets up for Watcom C++ 11.0 32-bit Windows compilation. Substitute this for any of the batch files in the bin directory for the compiler you are using. You can choose from the following batch files to configure the build environment for different supported compilers:

<b>32-bit DOS protected mode support:</b>	
bc45-d32.bat	Borland C++ 4.52 (DPMI32)
bc50-d32.bat	Borland C++ 5.0 (DPMI32)
bc50-d32.bat	Borland C++ Builder 5.0 (DPMI32)
gcc2-dos.bat	GNU C++ 2.9.5 (DJGPP 2.02)
w10ad32.bat	Watcom C++ 10.0a (DOS4GW)
wc10-d32.bat	Watcom C++ 10.6 (DOS4GW)
wc11-d32.bat	Watcom C++ 11.0 (DOS4GW)

<b>32-bit Windows GUI support:</b>	
bc45-w32.bat	Borland C++ 4.52 GUI programs
bc50-w32.bat	Borland C++ 5.0 GUI programs
bc50-w32.bat	Borland C++ Builder 5.0 GUI programs
gcc2-w32.bat	GNU C++ for Win32 (Cygwin) GUI programs
vc40-w32.bat	Microsoft Visual C++ 4.2 GUI programs
vc50-w32.bat	Microsoft Visual C++ 5.0 GUI programs
Vc60-w32.bat	Microsoft Visual C++ 6.0 GUI programs
wc10aw32.bat	Watcom C++ 10.0a GUI programs
wc10-w32.bat	Watcom C++ 10.6 GUI programs
wc11-w32.bat	Watcom C++ 11.0 GUI programs

<b>32-bit Windows console support:</b>	
bc45-c32.bat	Borland C++ 4.52 console programs
bc50-c32.bat	Borland C++ 5.0 console programs
bc50-c32.bat	Borland C++ Builder 5.0 console programs
gcc2-c32.bat	GNU C++ for Win32 (Cygwin) console programs
vc40-c32.bat	Microsoft Visual C++ 4.2 console programs
vc50-c32.bat	Microsoft Visual C++ 5.0 console programs
Vc60-c32.bat	Microsoft Visual C++ 6.0 console programs
wc10aw32.bat	Watcom C++ 10.0a console programs
wc10-c32.bat	Watcom C++ 10.6 console programs
wc11-c32.bat	Watcom C++ 11.0 console programs

Note also that once you have properly set up the makefile utilities, you can switch between different compilers from the command line simply by calling one of the above

batch files. This makes it easy to test and compile your own code with multiple compilers on a single machine.

For Windows development, you may also wish to note that the difference between console and GUI compilation is controlled by the WIN32\_GUI environment variable. If you find that compiling SDK examples cannot be completed due to missing references to WinMain or DEF definition file, it is probably because that example is simply a console-only application. So try re-compiling with the "c32" version batch file instead of "w32".

### Win32 hosted tools (start-sdk.bat)

The Win32 hosted tools are identical to the 32-bit DOS hosted tools, however they are native Win32 tools and hence can support long filenames properly under Windows 9x and Windows NT/2000/XP. The Win32 tools usually runs slower than the DOS hosted tools on Windows 9x, but significantly faster on Windows NT/2000/XP. Unless you need long filenames, you may want to stick with the DOS hosted tools for Windows 9x environments. By default the startup scripts will automatically detect the host platform and use the Win32 tools for Windows NT/2000/XP and the DOS tools for Windows 9x.

### Windows hosted tools for RTTarget-32 (start-sdk.bat)

In order to use the Windows hosted environment for On Time RTTarget-32 development, you need to additionally modify the `bin\set-vars.bat` batch file to reference the location where you have installed the RTTarget-32 SDK headers and libraries.

Once you have the startup file configured, you then need to run the following each time you start a command shell to enable the SciTech makefile utilities (a good idea to put into your startup batch files):

```
call c:\scitech\bin\set-vars.bat
call c:\scitech\bin\vc60-rtt.bat
```

The second batch file sets up the compiler configuration for your default compiler. The line above sets up for Microsoft Visual C++ 6.0 32-bit RTTarget-32 compilation. You can choose from the following batch files to configure the build environment for different supported compilers:

32-bit RTTarget-32 support:	
bc50-rtt.bat	Borland C++ 5.0 RTTarget-32 programs
bc5-rtt.bat	Borland C++ Builder 5.0 RTTarget-32 programs
vc50-rtt.bat	Microsoft Visual C++ 5.0 RTTarget-32 programs
vc60-rtt.bat	Microsoft Visual C++ 6.0 RTTarget-32 programs
ow10-rtt.bat	Open Watcom C++ 1.x RTTarget-32 programs

Note that invoking these RTTarget-32 batch files may set or clear options for linking with additional On Time libraries like RTFiles-32 and RTKernel-32, or the DLL version of RTTarget-32. You can change the environment variable options from the command line, though may wish to change them by modifying the respective batch files.



Also note that while On Time RTTarget-32 version 4.x release does not officially support Watcom C, it is still possible to compile and link with the DLL version of RTTarget-32 using OpenWatcom C 1.x and the DLL import libraries provided in the On Time SDK for Microsoft Visual C.

---

*Note: If you wish to use RTFiles-32 filesystem support (controlled by the USE\_RTFILES32 environment variable), you will need to copy the RTFiles-32 SDK example Init.c module or similar version into the PM library compilation path. This will insure that the RTFiles-32 filesystem declarations are included in your SNAP application and initialized at startup by RTTarget-32. If your own custom modified version renamed the initialization function, you should refer to the PM library source to make any changes.*

---

## OS/2 hosted tools (start-sdk.cmd)

The first thing you need to do is edit the `bin-os2\set-vars.cmd` script file to reference the location where you have installed the files, and the locations where all your compilers are installed. See the comments in `set-vars.cmd` for more information.

Once you have the startup file configured, you then need to run the following each time you start a command shell to enable the SciTech makefile utilities (a good idea to put into your startup batch files):

```
call c:\scitech\bin-os2\set-vars.cmd
call c:\scitech\bin-os2\wc11-o32.cmd
```

The second batch file sets up the compiler configuration for your default compiler. The line above sets up for Watcom C++ 11.0 32-bit OS/2 compilation. Substitute this for any of the batch files in the bin directory for the compiler you are using. You can choose from the following batch files to configure the build environment for different supported compilers:

<b>32-bit OS/2 GUI support:</b>	
<code>emx-p32.cmd</code>	GNU C++ (emx) Presentation Manager programs
<code>va30-p32.cmd</code>	IBM VisualAge for C++ 3.0 Presentation Manager programs
<code>va36-p32.cmd</code>	IBM VisualAge for C++ 3.65 Presentation Manager programs
<code>wc10-w32.cmd</code>	Watcom C++ 10.6 GUI Presentation Manager programs
<code>wc11-w32.cmd</code>	Watcom C++ 11.0 GUI Presentation Manager programs

<b>32-bit OS/2 console support:</b>	
<code>emx-o32.cmd</code>	GNU C++ (emx) console programs
<code>va30-o32.cmd</code>	IBM VisualAge for C++ 3.0 console programs
<code>va36-o32.cmd</code>	IBM VisualAge for C++ 3.65 console programs
<code>wc10-w32.cmd</code>	Watcom C++ 10.6 GUI console programs
<code>wc11-w32.cmd</code>	Watcom C++ 11.0 GUI console programs

Note also that once you have properly set up the makefile utilities, you can switch between different compilers from the command line simply by calling one of the above batch files. This makes it easy to test and compile your own code with multiple compilers on a single machine.

---

*Note: In order for any of the OS/2 sample programs to run, you must have the SDDHELP.SYS device driver loaded. Hence ensure you add this device driver to your CONFIG.SYS file and then reboot in order to start using the SNAP SDK programs. This file is also distributed along with the old SciTech Display Doctor for OS/2 as well as the more recent SciTech SNAP Graphics for OS/2 products.*

---

### Linux hosted tools (start-sdk.linux)

The first thing you need to do is edit the `bin/set-vars-linux.sh` script file to reference the location where you have installed the files, and the locations where all your compilers are installed. See the comments in `set-vars-linux.sh` for more information.

Once you have the startup file configured, you then need to run the following each time you start a command shell to enable the SciTech makefile utilities (a good idea to put into your startup scripts such as `.bash_profile`):

```
. ~/scitech/bin/set-vars-linux.sh
. ~/scitech/bin/gcc-linux.sh
```

The second script file sets up the compiler configuration for your default compiler. The line above sets up for GNU C/C++ for Linux (eventually other compilers will be supported such as Borland Kylix).

Note that in the above script code, you need to **source** the script files to ensure the environment variables are exported to your regular shell. Hence make sure the leading `!` is included!

If you are developing on an older libc5 based system (as opposed to the newer glibc as used by Red Hat 5.x and later), you will also need to set the `LIBC=1` environment variable. This will tell our script files that you are running on an older system and need to use the libc5 compiled binaries, and to put your compiled libraries into the libc5 directories.

Also in order to build the SNAP libraries, you should have the latest Linux 2.x kernel sources installed. The PM library depends upon a number of header files from the 2.x kernels (`joystick.h` and `mtsr.h`) in order to build. You can build for older kernels if you wish, but you will need to modify the PM library `makefile` to do this.

### QNX hosted tools (start-sdk.qnx)

The first thing you need to do is edit the `bin/set-vars-qnx.sh` script file to reference the location where you have installed the files, and the locations where all your compilers are installed. See the comments in `set-vars-qnx.sh` for more information.

Once you have the startup file configured, you then need to run the following each time you start a command shell to enable the SciTech makefile utilities (a good idea to put into your startup scripts such as `.bash_profile`):

```
. ~/scitech/bin/set-vars-qnx.sh
. ~/scitech/bin/qnx4.sh
```

The second script file sets up the compiler configuration for your default compiler. The line above sets up for Watcom C++ 10.6 for QNX.

Note that in the above script code, you need the *source* the script files to ensure the environment variables are exported to your regular shell. Hence make sure the leading `!` is included!

Note also that for QNX development you will need to set the `USE_BIOS=1` environment variable to enable support for calling the BIOS. You will also need to copy the `vbios.lib` files from the `drivers/qnx` directory into your runtime library directories. The final release will allow you to build SNAP programs without requiring the BIOS support, but for the moment the BIOS support is required to run under QNX, even though it is not used if you are running on SciTech SNAP drivers.

## Compiling SciTech SNAP Graphics

---

Once you have all the startup scripts configured and executed, you are ready to begin compiling. Building all the SNAP libraries in one fell swoop is very easy. Simply change into the `src` directory below where you have installed all the files and issue a `dmake build`. Ie:

```
cd scitech/src
dmake build
```

Using `dmake build` will force build all the libraries, and will build all the libraries with your selected compiler and using the currently configured options. If any errors are encountered during the build, it will stop and you can fix the errors and then restart the build from the offending library with a simple `dmake` command (ie: the default target builds for the selected compiler). You can also build each library from each directory if you wish as well.

By default the makefile utilities hide the actual command lines used to call the compiler on Unix systems to clean up the compile process output. This makes it much easier to see errors and warnings in the code as it compiles. If you need to see the actual command lines passed to the compiler, you can build using the `SHOW_ARGS=1` variable (ie: `dmake build SHOW_ARGS=1`).

### Compiling release and debug builds

We have also created script files to build all libraries for your selected compiler in both 'debug' and 'release' modes. To build the debug libraries, go into the `src` directory and issue run the `mkdebug` script files. Ie:

```
cd scitech/src
mkdebug
```

To build the release libraries, go into the `src` directory and issue run the `mkrelease` script files. Ie:

```
cd scitech/src
mkrelease
```

All debug libraries end up under the `lib/debug` directory tree, and all release libraries end up under the `lib/release` directory tree. To link your own programs with the debug libraries, you set the environment variable `CHECKED=1` (or set it on the `dmake` command line). This will tell the `dmake` makefile scripts to build in `CHECKED` debug mode (add extra runtime checks) as to get the libraries from the `lib/debug` directories. Otherwise the libraries are pulled from the `lib/release` directory.

## Compiling the sample programs

Once you have all the libraries built, you can try to compile some of the sample programs using `dmake`. Building a sample program is as simple as changing to the directory where the sample program exists, and issuing a `dmake` command. To build the `GATest` example program for instance, you would do the following:

```
cd %SCITECH%\examples\snap\graphics\gatest
dmake
```

## Setting up Your Compiler Configuration

Once you have installed the source code and compiled the libraries from the command line, you will need inform your compiler where the include files and library files are located for your own projects. The following steps provide a guide to setting things up correctly for your compiler.

After the files are installed, all include files into the `%SCITECH%\include` directory under the installation directory where you chose to install the product. So if you installed the product in `c:\scitech`, the include file directory would be `c:\scitech\include`). If you are compiling your applications from the Integrated Development Environment (IDE) for your compiler, you will need to set the include directories for your project file's to include the `%SCITECH%\include` directory.

If you are compiling from the command line, you simply need to add the `%SCITECH%\include` directory to your `INCLUDE` path environment variable (or the command line configuration file for your compiler if it doesn't use environment variables).

When the libraries are built, the library files end up under the `%SCITECH%\lib` directory under the installation directory where you chose to install the product. So if you installed the product in `c:\scitech`, the library directory is `c:\scitech\lib`). Beneath this directory is are `debug` and `release` directories, containing debug and release versions of the libraries (depending on what you compiled or installed). Beneath those two directories is a hierarchy of directories containing library files for different

operating systems and different compilers as shown in the tables below (there may be more if there are more compilers supported in a particular release):

<b>32-bit DOS protected mode support:</b>	
dos32\bc4	Borland C++ 4.52 32-bit DOS libraries
dos32\bc5	Borland C++ 5.0 32-bit DOS libraries
dos32\wc10	Watcom C++ 10.6 32-bit DOS libraries
dos32\wc11	Watcom C++ 11.0 32-bit DOS libraries
dos32\dj2	DJGPP 2.01 32-bit DOS libraries

<b>32-bit Windows:</b>	
win32\bc4	Borland C++ 4.52 32-bit Windows libraries
win32\bc5	Borland C++ 5.0 32-bit Windows libraries
win32\bc5	Borland C++ Builder 5.0 32-bit Windows libraries
win32\vc40	Microsoft Visual C++ 4.0 32-bit Windows libraries
win32\vc50	Microsoft Visual C++ 5.0 32-bit Windows libraries
win32\vc60	Microsoft Visual C++ 6.0 32-bit Windows libraries
win32\wc10	Watcom C++ 10.6 32-bit Windows libraries
win32\wc11	Watcom C++ 11.0 32-bit Windows libraries

<b>32-bit RTTarget-32:</b>	
rtt32\bc50	Borland C++ 5.0 32-bit RTTarget-32 libraries
rtt32\bc5	Borland C++ Builder 5.0 32-bit RTTarget-32 libraries
rtt32\vc50	Microsoft Visual C++ 5.0 32-bit RTTarget-32 libraries
rtt32\vc60	Microsoft Visual C++ 6.0 32-bit RTTarget-32 libraries
rtt32\ow10	Open Watcom C++ 1.x 32-bit RTTarget-32 libraries

<b>32-bit OS/2:</b>	
os232\emx	GNU C++ for OS/2 libraries (emx)
os232\va3	IBM VisualAge for C++ 3.0 OS/2 libraries
os232\va36	IBM VisualAge for C++ 3.65 OS/2 libraries
os232\wc11	Watcom C++ 11.0 32-bit OS/2 libraries
os232\ow10	Open Watcom C++ 1.x 32-bit OS/2 libraries

<b>Linux:</b>	
linux/gcc/x86/a	GNU C++ for Linux static libraries (x86)
linux/gcc/x86/so	GNU C++ for Linux shared libraries (x86)
linux/gcc/ppc-be/a	GNU C++ for Linux static libraries (PowerPC Big Endian)
linux/gcc/ppc-be/so	GNU C++ for Linux shared libraries (PowerPC Big Endian)
linux/gcc/alpha/a	GNU C++ for Linux static libraries (Alpha Big Endian)
linux/gcc/alpha/so	GNU C++ for Linux shared libraries (Alpha Big Endian)

<b>QNX:</b>	
qnx4/wc10	Watcom C++ 10.6 for QNX 4
qnx4/wc11	Watcom C++ 11.0 for QNX 4
Qnxnto	GNU C++ for QNX Neutrino

If you are compiling your applications from the IDE for your compiler, you will need to set the library directories for your project file to include the %SCITECH%\lib\... directory (select the appropriate directory from Tables above). If you are compiling from the command line, you simply need to add the %SCITECH%\lib\... path to your LIB path environment variable (or the command line configuration file for your compiler if it doesn't use environment variables).

Once you have done the above steps, you should then be able to compile and link your own programs using the SDK.

---

*Note: For Watcom C++ users, by default Watcom C++ compiles all source code using register based parameter passing. Hence by default all SciTech Software libraries are compiled with register based parameter passing. If you are compiling and linking you code for stack based parameter passing, you will need to link with a different set of libraries. All libraries can be compiled with either stack and register based calling. The stack based libraries will have the same name as the register based versions of the libraries, but will have an extra 's' added to the front of the library name. To build or link to the stack call libraries, use the STKCALL=1 option to dmake.*

---

## Using the Makefile Utilities

---

Once you are up and running with your default compiler configuration, you should be able to run dmake from the directory containing the sample programs or libraries that you wish to compile. If things run smoothly you should get a resulting executable file or library. The section describes some of the more common options you can use to control the compile environment using dmake.

### Standard Makefile Targets

All of the makefile utilities startup scripts support a standard set of targets for controlling the compilation for the current compiler. The most common commands and

useful targets that you may want to use when building examples and re-compiling any libraries are listed in the table below:

<code>dmake</code>	Running <code>dmake</code> by itself in a directory will select the default target for the makefile, which is usually to compile and link all sample programs in that directory. Some makefiles only support building libraries so the default target may produce a library rather than an executable file.
<code>dmake -u</code>	The <code>-u</code> command line option forces a complete re-build of all files, so it is useful to re-build an entire directory from scratch.
<code>dmake lib</code>	This builds just the library for the directory.
<code>dmake install</code>	This builds just the library for the directory and then installs the library into the appropriate <code>c:\scitech\LIB\xxx\xx</code> directory. You should only do this once you are sure that everything is working correctly! The new library will overwrite the old library.
<code>dmake clean</code>	This cleans out all object files, libraries and pre-compiled header files etc from the directory, but leaves all executable files and shared libraries.
<code>dmake cleanexe</code>	This cleans out all non-source files including all executeable files and shared libraries.

### Standard Makefile Options

All of the makefile utilities startup scripts support a standard set of options for controlling the way that the compilation is performed. Makefile options are provided for turning on debug information, speed or size optimizations and inline floating point instructions. By default when you build files, no optimizations and no debugging information is generated. The following table lists the most common and useful of these options for building examples and re-compiling any libraries.

DBG	Turns on debug information
OPT	Turns on speed optimizations
OPT_SIZE	Turns on size optimizations
FPU	Turns on inline floating point arithmetic
BUILD_DLL	Build a dynamic link library of Unix shared library
STKCALL	Turns on stack calling conventions for Watcom C++
SHOW_ARGS	Show full arguments passed to compiler under Unix
CHECKED	Compile and link against checked debug libraries
MAX_WARN	Turn on maximum compiler warning setting
USE_PMODEW	Use PMODE/W DOS extender for 32-bit DOS
USE_CAUSEWAY	Use CauseWay DOS extender for 32-bit DOS

All of the above options can be passed to dmake in one of two ways: on the command line or as global environment variables. For instance the following are equivalent:

```
dmake DBG=1 OPT=1 install
```

or

```
set DBG=1
set OPT=1
dmake install
```

The primary difference between the two is that by setting the environment variables, you change the default behavior for dmake, so that every time you build something those options will be in effect. The environment variable mechanism is useful to set the most common options that you will use so you don't have to constantly pass them on the command line.

## CauseWay DOS Extender Support

---

The SciTech SNAP Graphics SDK is designed to work with Open Watcom C++ and both the DOS4G/W and CauseWay DOS extenders. It is recommended that you use the latest versions, as included with Open Watcom 1.2 or later. Earlier versions may not support required features, such as Intel MTRR programming.

In order to compile and link your programs to use CauseWay, either set the `USE_CAUSEWAY=1` environment variable, or pass this variable on the command line to the dmake program.

## Connecting with Perforce

---

SciTech Software uses the commercial Perforce version management software to maintain its public Open Source software repository. Perforce Software has graciously allowed Open Source software projects to be able to use a free server license for that source code. This section details how you can use Perforce to keep up to date with the absolute latest versions of the SciTech SNAP SDK.

### Download a Perforce Client

In order to get Perforce up and running on your system, you first need to download the Perforce client program for your Operating System. Perforce Software provides clients for just about every OS freely on their web site at:

<http://www.perforce.com/perforce/loadprog.html>

All you need to do is download the appropriate p4 binary for your Operating System from their web site. You may also want to download or browse the documentation for p4, although it is quite simple to use from the command line and has extensive command line help.



---

*Note: If you have not upgraded recently you should upgrade your copy of Perforce to 2002.1 or later, which has some new features that our Perforce server supports. However our depot should work with most Perforce clients right back to the 97.x series.*

---

In order to use Perforce you will need a TCP/IP connection to the internet, either via modem or direct connection. Accessing Perforce via modem is actually quite fast, and we regularly have developers doing remote development over slow modem links from around the world.

## Setting up your environment for anonymous access

Once you have downloaded the p4 binary and put it on your path, you want to set the following environment variables:

```
P4PORT    = perforce.scitechsoft.com:3488
P4USER    = anonymous
P4PASSWD  = anonymous
P4CLIENT  = YOURNAME_YOURMACHINE_YOUROS
P4EDITOR  = (path to your favorite console editor, such as vi)
```

where you would replace YOURNAME\_YOURMACHINE\_YOUROS with a unique name (you can't use a client that someone else is already using anonymously!). For instance a valid client name might be: JOEBLOGGS\_DEVEL\_LINUX.

The client names the view that maintains the information about what files you have checked out on that system, and it must be unique for each user, and each machine. If you are doing development under multiple Operating Systems or multiple machines, you need a separate client view for each machine and each Operating System that will contain a copy of the source code from Perforce (hence the above naming convention for clients).

Once you have the above variables set up, you can see if Perforce can access our server by typing the following:

```
p4 info
```

You should see something similar to the following if it can connect successfully:

```
User name: anonymous
Client name: JOEBLOGGS_DEVEL_LINUX
Client host: MYMACHINE
Client root: c:\
Current directory: c:\scitech\src
Client address: 65.209.3.29:3643
Server address: perforce.scitechsoft.com:3488
Server root: /home/perforce
Server date: 2000/12/17 16:21:08 PST
Server version: P4D/LINUX52X86/2000.1/17250 (2000/09/11)
Server license: Scitech Software 58 users (support ends 2001/06/21)
```

## Setting up your client mapping

Once you are able to connect, you need to set up your client mapping since by default it will be empty. The client space is what allows you to map in only the portions of the

software repository that you care about, and where to put them on your local system. The client information is then stored remotely on our server so it knows what files you have on your system and where they live (so if you delete stuff manually you will need to do a forced sync!).

To set up your client mapping, type:

```
p4 client
```

which will bring you into your editor of choice. It will include comments at the top about what parts of the file are used for what. There are two parts that you need to modify; the 'root' definition and the 'view' definition. The 'root' field is the root directory where all files get checked out onto your system, and this should be the root of the scitech directory (ie: /home/KendallB/scitech is where I put the sources on Unix systems, or c:\scitech for DOS, Windows and OS/2 systems). For example:

```
Root:      /home/KendallB/scitech
```

The 'options' field defines what options are in effect for that client. The most important one you will want to change is the 'nocompress' option to 'compress'. By enabling compression, you will drastically reduce the time required for syncs over an internet connection! The second most useful option is to change the 'normdir' option to 'rmdir'. This will have Perforce remove empty directories from the file system when all the files managed by Perforce are deleted and the directory is empty.

The 'view' field defines what parts of the perforce repository you want mapped to your local drive. The simplest mapping will pull down all the source files you would need (and the one you should probably use) would be:

```
View:      //depot/gpl/... //YOURNAME_YOURMACHINE_YOUROS/...
```

where you would replace the YOURNAME\_YOURMACHINE\_YOUROS above with the name of your client. This line indicates that all files and directories get mapped to the root of the client (or /home/KendallB/scitech if you uses the above root field). Now once you have the client view defined, exit your editor and it will send the changes to the perforce server.

## Syncing up for the first time

The SciTech Perforce software repository contains a lot of code, and doing a full sync will pull down well in excess of 60Mb of files! Hence in order to save time for the initial sync, you can download and the latest 'full depot' archive to your system, and then tell Perforce that you have all files at the time that release was made. Then when you sync up you will only pull down all the files that have changed since the version of the full depot files you downloaded. You will need to first download one of the following files depending on the platform you are using:

full_depot_r13.zip	Complete copy of GPL depot in DOS/Win format
full_depot_r13.tar.gz	Complete copy of GPL depot in Unix format

Once you have the full depot files downloaded and installed, you can do the initial sync for using the following commands:

```
p4 flush -f @public_release_13
p4 sync
```

This will tell Perforce that you have all the files at the label 'public\_release\_13'. You can change this label to the appropriate label for the public release that you have downloaded. Then the 'p4 sync' command grabs any changes that have occurred since the 'public\_release\_13' label. Once you have synced up the first time, you should only need to use 'p4 sync' to get the latest updates to all the files from that point forward.

## Using Perforce from the command line

Using Perforce is very easy, and you can get help from the command line with:

```
p4 help
```

The most common commands you might need to use are as follows.

```
p4 sync      - Sync up with the latest sources
p4 edit      - Open a file for editing
p4 add       - Add a new file to the repository
p4 delete    - Delete a file from the repository
p4 resolve   - Resolve conflicts if two people edit the same file
p4 revert    - Revert changes for a currently opened file
p4 submit    - Submit all changes to the repository
p4 opened    - List all files you currently have opened
```

The big difference between Perforce and other SCM systems like RCS is that Perforce is 'change' oriented. When you submit a change list, included in the change list will be all of the files that you currently have opened. If you don't want a file in that change list, simply delete it from the change list in the editor and it won't be submitted but will remain in your list of open files after the submit. You *must* provide a description for the change, and then the entire set of files associated with the change is submitted as a single atomic change to the server, including file additions and deletions! Hence when you need to revert back to older versions of the code, you can do it via the revision number for a specific file, a change list number to go back to the state when a particular change was submitted, or a label defined earlier.

Note that anonymous users only have read access to the software repository, and will not be able to open any files or submit changes. If you wish to be able to contribute changes to the SciTech Perforce repository, please contact [perforce@scitechsoft.com](mailto:perforce@scitechsoft.com) directly to find out how you can get a developer account with read/write access.

# Programming with SNAP Graphics

This chapter describes in detail the issues application and system software developers will face when developing code to use the SciTech SNAP, Graphics Architecture.

## Loading and Initializing SciTech SNAP Graphics

This section contains an overview of the procedures required to load and initialize the SciTech SNAP Graphics Architecture drivers.

### Runtime Library Standard Locations

In order to be able to use the SciTech SNAP Graphics Architecture, the library must be dynamically loaded and initialized with a driver specific to the installed hardware loaded into memory. The SciTech SNAP Graphics Architecture interface library (n\_ga.lib) contains the code necessary to load and initialize the Binary Portable DLL, and provide access to the internal functions. This library is provided in source code form and has support for popular operating systems and compilers. All the loader library code is pure ANSI C code, so should be portable to any operating system and CPU family.

Drivers for the SciTech SNAP Graphics Architecture are generally located in the following standard locations on the end users system:

Operating	Default Directory Location
MSDOS	c:\snap\drivers
Windows 95/98	c:\windows\system\snap
Windows NT	c:\winnt\system32\snap
OS/2	c:\os2\drivers\snap
Linux	/usr/lib/snap
QNX 4.x	/qnx4/snap/bin
QNX 6.x	/nto/snap/<arch>/bin

In many cases the above directories are relative to the operating system root directory, so the drive letters may be different depending on how the operating system was installed. Note also that the user may set the SNAP\_PATH environment variable and the loader library will check in this directory for the driver file(s). If the driver cannot be found in the global device driver directory, the loader library will also look in the current directory so that applications may ship OEM versions of the drivers local to their application.

Note also that the locations for the Windows and OS/2 operating systems are in the standard device driver directories, and may not necessarily be located in the c:\windows or c:\os2 directories.

For On Time RTTarget-32 target system without RTFiles-32 filesystem support, the SNAP Graphics BPD files will be bound into the RTB binary image, so it is entirely

optional whether you use file paths or not. Refer to the example RTTarget-32 configuration files provided with SNAP and MGL demos courtesy of On Time Informatik.

## Enumerating Installed Devices and Loading a Driver

Before you can use the SciTech SNAP Graphics Architecture library, you must first enumerate all the devices in the system and load a device driver for the selected device that you wish to control. Enumerating the devices in the system is done using the *GA\_enumerateDevices* function. This function returns a count of the number of display devices installed in the system. Then you call *GA\_loadDriver* to load and initialize a valid device driver for the device you wish to control. The primary display device is always device index 0, and the first secondary controller is device index 1, the next is device index 2 and so on. When you are finished you must then call *GA\_unloadDriver* to unload the device driver from memory. If you only need to support the primary display controller, you can skip the call to *GA\_enumerateDevices* and simply call *GA\_loadDriver* with a device index of 0. For example:

```
void main(void)
{
    GA_devCtx *dc;

    if ((dc = GA_loadDriver(0)) == NULL)
        PM_fatalError("Unable to load graphics driver!");
    ... do some stuff with the driver
    GA_unloadDriver(dc);
}
```

## Locating and Calling Device Driver Functions

Once you have a pointer to the device driver context structure, you then need to locate the pointers to the device driver functions of interest. Once a graphics device driver is loaded, the *GA\_queryFunctions* function is the single entry point where you can query the driver to determine what functions it supports and to obtain pointers to those functions. All of the functions in the device driver are grouped together into functional groups, and you can obtain pointers to each individual group of functions by calling *GA\_queryFunctions* with the identifier of the group you are interested in. If the driver does not support a particular group of functions, it may return FALSE, indicating the hardware device does not support that entire function group. Within a particular group of functions, certain entry points may be set to NULL if the driver or hardware does not support that particular feature within the group.

For example in order to call the *GetVideoModeInfo* function to enumerate the available display modes, you need to fill in the structure of pointers to the main init functions. To do this, use the following code (make sure you fill in the *dwSize* member with the size of the structure you are passing in!):

```
GA_initFuncs initFuncs;

bool LoadInitFuncs(GA_devCtx *dc)
{
    initFuncs.dwSize = sizeof(initFuncs);
    if (!GA_queryFunctions(dc, GA_GET_INITFUNCS, &initFuncs))
```

```

        return false;
    return true;
}

```

Once you have the list of function pointer for the function group you are interested in, you can simply call the returned functions via the function pointers contained within the structure. Make sure you first check that the function pointer is not NULL, since it is valid for optional functions to be returned as NULL by the loaded drivers.

---

*Note: You **must** obtain a copy of all function groups (except the GA\_initFuncs group) every time that you change display modes, as the features and capabilities of the driver and/or hardware device may change between different screen resolutions and color depths.*

---

## Querying Device Configuration Information

Once you have a driver loaded and initialized, it is sometimes useful to inform the user what type of device was detected. Once you have a pointer to the device driver initialization function group, you can use the *GetConfigInfo* function to get complete information about the manufacturer of the installed device, the name of the device as well as information about when the device driver was tested and certified. All the information is returned in the *GA\_configInfo* structure, and can be obtained as follows:

```

ibool GetConfigInfo(GA_devCtx *dc)
{
    GA_initFuncs  initFuncs;
    GA_configInfo info;

    initFuncs.dwSize = sizeof(initFuncs);
    if (!GA_queryFunctions(dc, GA_GET_INITFUNCS, &initFuncs))
        return false;
    info.dwSize = sizeof(info);
    init.GetConfigInfo(&info);
    // do something useful with the information!
    return true;
}

```

---

*Note: Make sure you initialize the dwSize member of the GA\_configInfo structure to the size of the structure being passed to GetConfigInfo!*

---

## Working With Display Modes

This section discusses how to determine what display modes are supported by the installed device, how to change refresh rates and how to update the list of available display modes.

### Finding Available Display Modes

Before any of the drawing functions can be called, one of the supported display modes must be initialized by the application program by calling the *SetVideoMode* function. The SciTech SNAP Graphics Architecture does not define a standard display mode numbering scheme but rather relies on the application to search through the list of available display modes for one that has the desired resolution and pixel depth. In order to find a valid video mode number to be passed to *SetVideoMode*, the *GetVideoModeInfo*

function is used to obtain specific information about all of the available video modes supported by the loaded driver. The list of available video modes is stored in the `AvailableModes` field of the `GA_devCtx` structure. Once the desired display mode has been identified, this display mode number can be used in the call to `SetVideoMode`.

The general procedure you would normally follow to find the identifier for a display mode with a particular X and Y resolution is as follows:

```
N_uint16 FindGraphicsMode(
    GA_devCtx *dc,
    int xRes,
    int yRes,
    int bitsPerPixel)
{
    GA_initFuncs    init;
    GA_modeInfo     modeInfo;
    N_uint16        *modes;

    /* Load the driver init functions */
    init.dwSize = sizeof(init);
    if (!GA_queryFunctions(dc, GA_GET_INITFUNCS, &init))
        return 0xFFFF;

    /* Search for the display mode */
    for (modes = dc->AvailableModes; *modes != 0xFFFF; modes++) {
        modeInfo.dwSize = sizeof(modeInfo);
        if (init.GetVideoModeInfo(*modes, &modeInfo) != 0)
            continue;
        if (modeInfo.Attributes & gaIsTextMode)
            continue;
        if (modeInfo.XResolution == xRes &&
            modeInfo.YResolution == yRes &&
            modeInfo.BitsPerPixel == bitsPerPixel)
            return *modes;
    }
    return 0xFFFF;
}
```

## Refresh Rate Control

Once you have found the identifier for the display mode you wish to use, you can then call the `SetVideoMode` function to set the display mode. One of the parameters to the `SetVideoMode` function is the refresh rate to be used for the mode. In most normal situations you would pass a value of 0 for this parameter, which will set the currently selected default refresh rate for the mode. However it is possible for the application to directly specify the refresh rate to be used via two different methods.

The first method is to pass the desired refresh rate to the `SetVideoMode` function as a specific value in Hz (ie: the integer value 60 for 60Hz etc). However this will only work if the refresh rate requested is actually supported by the display mode, so you need to first determine what refresh rates are supported. One of the bits of information returned by the `GetVideoModeInfo` function, is a complete list of all refresh rates for that display mode. This list of refresh rates will be properly filtered based on the capabilities of the underlying graphics hardware and the attached monitor (assuming a monitor has been selected by the user or detected automatically by Plug and Play). If you pass a value that

is not listed in the information block returned by the *GetVideoModeInfo* function, the *SetVideoMode* function will return -1, indicating failure.

The second method allows for refresh rates that are not listed as part of the drivers standard timings, such that an application can provide an exact set of CRTC timings to be used for the display mode (useful for flat panels and fixed frequency displays). Alternatively if the application just needs a non-standard, custom refresh rate, the application can use the VESA Generalised Timing Formula (GTF) functions provided to compute a set of generic CRTC timings for any arbitrary display mode and refresh rate (see *GA\_computeCRTCtimings*). Once the application has used the GTF functions to compute a set of CRTC timings for the display mode, those CRTC timings can then be passed directly to the *SetVideoMode* function along with use of the *gaRefreshControl* mode flag. This second method provides for the maximum versatility and support for specialised applications (such as fixed frequency monitors, LC stereo shutter glasses and head mounted displays).

## Using Custom Display Modes

Although the SciTech SNAP Graphics Architecture exports a list of supported display modes, internally there is actually no real concept of a standard display mode. Instead SciTech SNAP Graphics maintains a profile of known display modes, and uses that profile to enumerate to applications what display modes are available. The profile of known display modes also includes supported refresh rates, and that information is used to create the *GA\_modeInfo* information returned by the *GetVideoModeInfo* function. When the application calls the regular *SetVideoMode* function, internally the drivers end up calling the *SetCustomVideoMode* function with the resolution and CRTC timings for the known display mode taken from the mode profile.

Sometimes special applications may require the need to set a custom display mode that is not listed in the regular profile of known display modes. To set a completely custom display mode, you can call the *SetCustomVideoMode* function directly and pass in your own set of CRTC timings that match the display mode of choice (use *GA\_computeCRTCtimings* to compute the CRTC timings with the GTF formulas). Note however that not all resolutions can be supported by all hardware (for instance some hardware requires the X resolution to be on 8 pixel boundaries, some 16 pixel boundaries). In order to determine if a desired custom resolution is actually supported by the hardware, you should call the *GetCustomVideoModeInfo* function. This function will properly round up the resolution parameters as necessary to support the mode on the underlying hardware.

---

*Note: You can also add a custom display mode to the default mode profile as an alternative to always calling the *SetCustomVideoMode* function.*

---

## 2D Coordinate System

All the device driver accelerator functions take coordinates in a local framebuffer coordinate system, which is established with a call to the *SetDrawBuffer* function. The coordinate system starts with (0,0) at the start of the active drawing buffer and



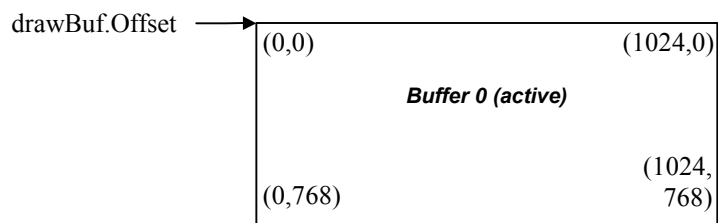
increments the X coordinate for every pixel and Y coordinate for every scanline. For instance in an 8-bit display mode, if the logical scanline width is set to 1024 bytes and the drawing buffer offset is set to 0, then the coordinate (0,1) will be rendering into the byte at location 1024 from the start of framebuffer memory. It is then up to the application to impose any other logical coordinate system on top of the graphics device driver routines, such as handling viewport mapping etc. Also note that clipping is generally not implemented by most of the drawing functions, so all drawing must be clipped by the application code in software before calling the low level device driver code (some functions do however provide clipping support where doing clipping in software prior to calling the function can be expensive compared to doing it in the hardware drivers).

The *SetDrawBuffer* function **must** be called to initialise the active drawing buffer for all subsequent rendering functions, before any drawing takes place. It takes as a parameter a structure which defines the offset, pitch, width and height of the drawing buffer to be used in video memory. In general you will simply set the draw buffer offset to 0, the pitch to the value returned in the *bytesPerLine* parameter from the *SetVideoMode* function and the width and height to the dimensions of the display mode. For example the following code can be used to initialise the drawing buffer for single buffered environments:

```
GA_modeInfo modeInfo;
GA_buffer drawBuf;

... assume modeInfo has been filled in ...
drawBuf.dwSize = sizeof(drawBuf);
drawBuf.Offset = 0;
drawBuf.Stride = modeInfo.BytesPerScanLine;
drawBuf.Width = modeInfo.XResolution;
drawBuf.Height = modeInfo.YResolution;
if (state2d.SetDrawBuffer(&drawBuf) != 0)
    PM_fatalError("Unable to set draw buffer!");
```

For a display mode of 1024x768 this would result in the logical coordinate system similar to the following:




---

*Note: You must be careful when calling the *SetDrawBuffer* function because some hardware has special restrictions on the starting offset and scanline stride values for offscreen video memory buffers. To simplify offscreen memory management, please use the buffer manager functions provided by the *GA\_bufferFuncs* function group to create and manage all flip buffers and offscreen buffers, which automatically account for these hardware requirements.*

---

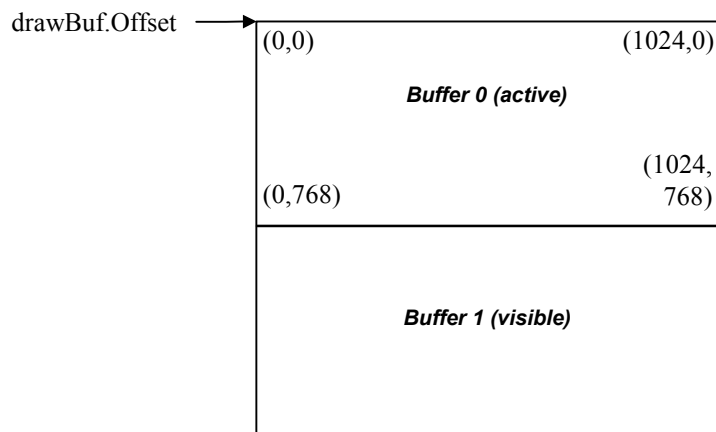
## Multi Buffering

Multi buffering can be achieved by using the *SetDrawBuffer* function to draw to a different location in video memory combined with the *SetDisplayStart* function to change the display start address to a different location in video memory. It is up to the application to determine how to divide up the display memory into the multiple display buffers, but for multi-buffering it is recommended that the display buffer offset always be aligned to a multiple of the scanline width for the display mode for maximum compatibility.

In the examples below we examine the case of multi buffering using two buffers, which is usually called double buffering. If enough offscreen video memory is available, multi buffering with more than 2 buffers can be useful because it allows applications to draw continuously without waiting for the vertical retrace when swapping the currently active visible buffer. For example:

```
activePage      = 0;
visiblePage     = 1;
drawBuf.dwSize  = sizeof(drawBuf);
drawBuf.Offset  = modeInfo.BytesPerScanLine *
                 (modeInfo.YResolution * activePage);
drawBuf.Stride  = modeInfo.BytesPerScanLine;
drawBuf.Width   = modeInfo.XResolution;
drawBuf.Height  = modeInfo.YResolution;
if (state2d.SetDrawBuffer(&drawBuf) != 0)
    PM_fatalError("Unable to set draw buffer!");
if (driver.SetDisplayStart(
    modeInfo.BytesPerScanLine *
    (modeInfo.YResolution * visiblePage));
```

will change the logical framebuffer layout in memory to the following:

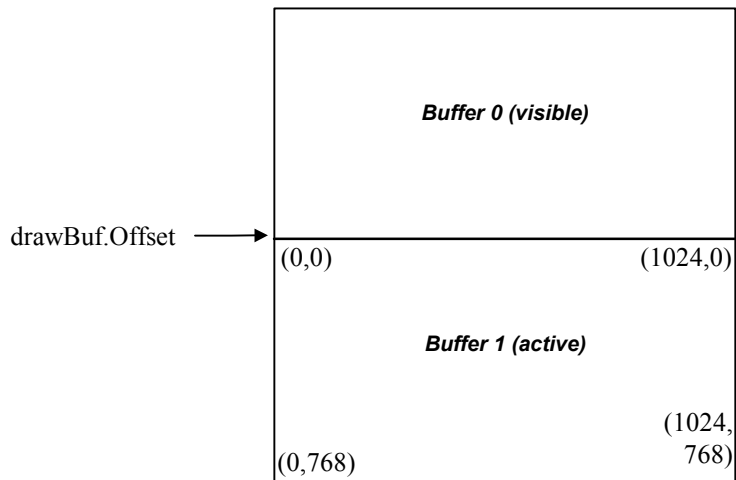


All drawing output is sent to the currently active buffer (buffer 0), and all video data is displayed from the currently visible buffer (buffer 1). Double buffering is achieved by using *SetDrawBuffer* to always draw to the hidden display buffer, and *SetDisplayStart* to make the CRT controller always display from a different buffer to the one that is currently being drawn into. The visible image can then be instantly updated by

swapping the new visible buffer to the buffer that was currently being rendered into with code similar to the following:

```
activePage      = 1;
visiblePage     = 0;
drawBuf.dwSize  = sizeof(drawBuf);
drawBuf.Offset  = modeInfo.BytesPerScanLine *
                  (modeInfo.YResolution * activePage);
drawBuf.Stride  = modeInfo.BytesPerScanLine;
drawBuf.Width   = modeInfo.XResolution;
drawBuf.Height  = modeInfo.YResolution;
if (state2d.SetDrawBuffer(&drawBuf) != 0)
    PM_fatalError("Unable to set draw buffer!");
if (driver.SetDisplayStart(
    modeInfo.BytesPerScanLine *
    (modeInfo.YResolution * visiblePage));
```

will change the logical framebuffer layout in memory to the following:




---

*Note: If you plan to utilize offscreen memory to store bitmap data, please use the buffer manager functions provided by the `GA_bufferFuncs` function group to create and manage all flip buffers and offscreen buffers. The buffer functions will provide for maximum compatibility across multiple hardware devices, properly accounting for hardware buffer alignment requirements for storing buffers in offscreen video memory.*

---

## Accessing Offscreen Video Memory

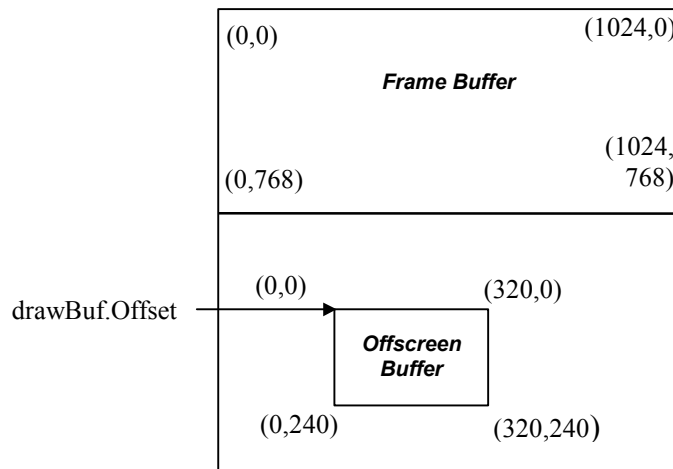
Offscreen video memory on the controller can be used for caching bitmap information to be used for fast *BitBlt* and *SrcTransBlt* operations for sprite animation. It can also be used to cache offscreen bitmap data for fast GUI operations. Drawing to offscreen display memory can be done in one of three ways. The first way is to simply use coordinates past the last Y coordinate for the currently active draw buffer, which is most useful when blitting cached bitmap data from offscreen video memory to display memory. The second way is to use the *SetDrawBuffer* function to make the offscreen memory the active draw buffer with the same dimensions as the main display screen. This will work on all controllers and is similar to multi-buffering mentioned above, but you never display from the offscreen buffer (note that you need to properly account for

the hardware alignment requirements for the offscreen buffer starting address and scanline pitch).

The third method is to use the *SetDrawBuffer* function to set the active drawing buffer to a non-conforming offscreen memory buffer on hardware that can support this. This is most useful for setting up offscreen 'surfaces' with a possibly different set of dimensions to the main display mode (similar to DirectX offscreen surfaces). The first two methods should be relatively straight forward, but the following code can be used to implement the third case for a 320x240 8bpp buffer starting at the 1Mb memory boundary:

```
drawBuf.dwSize = sizeof(drawBuf);
drawBuf.Offset = 1048576;    // 1Mb offset
drawBuf.Stride = 320;
drawBuf.Width = 320;
drawBuf.Height = 240;
if (state2d.SetDrawBuffer(&drawBuf) != 0)
    PM_fatalError("Unable to set draw buffer!");
```

Note that for non-conforming draw buffers, you *must* ensure that the Offset and Stride members are aligned to the necessary alignment values defined in the *BitmapStartAlign* and *BitmapStridePad* members of the *GA\_devCtx* structure (some hardware requires the use of the *AlignLinearBuffer* function instead to properly align the buffers). Also be prepared for *SetDrawBuffer* to fail if the hardware cannot do this, as lots of older hardware is not capable of supporting non-conforming draw buffers. The above code essentially sets up the following framebuffer layout:




---

*Note: If you plan to utilize offscreen memory to store bitmap data, please use the buffer manager functions provided by the *GA\_bufferFuncs* function group to create and manage all flip buffers and offscreen buffers. The buffer functions will provide for maximum compatibility across multiple hardware devices, properly accounting for hardware buffer alignment requirements for storing buffers in offscreen video memory.*

---

## Virtual Buffer Scrolling

Virtual scrolling functionality is selected by passing values other than -1 for the VirtualX and VirtualY parameters to the *SetVideoMode* function. By passing values other than -1 for the X and Y dimensions of the display mode, you will get a virtual mode such that the width or height of the mode is larger than the physically displayed image. Then you can use the *SetDisplayStart* function to change the display start address and scroll around within the display memory. The value passed to the *SetDisplayStart* function is a byte address in video memory for 8-bit and higher modes, so to move the display start address to the 10<sup>th</sup> pixel and 50<sup>th</sup> line you would use the following code:

```
if (driver.SetDisplayStart(
    modeInfo.BytesPerScanLine * 50 +
    10 * bytesPerPixel);
```

Don't forget that if you set a virtual display mode, that you must set the drawBuf.Stride value to the bytesPerLine value that was returned by the *SetVideoMode* function and not the value stored in the *GA\_modeInfo* structure for the mode! Some hardware devices have restrictions on the pitch that can be programmed for display modes, and this is taken into account when the driver initializes a virtual scrolling mode.

## Palette Programming During Double Buffering

If you wish to re-program the palette at the same time as performing double buffering (if the palette changes between frames during double buffering) then you should call the *SetDisplayStart* function first with the wait for retrace flag set, then *immediately* following you should program as many palette values as you can before the end of the retrace period. Note that on some older graphics devices and slower computers, you cannot program an entire set of 256 color lookup table entries during the retrace period before the onset of snow, so you will need to stagger the palette change over 2 or more retrace periods. Generally most devices can handle about 100-120 palette entries to be programmed per retrace (most newer hardware does not produce snow at all, so you can program all 256 at once).

## Integer Coordinates

Integer coordinates are passed as 32-bit signed integers to the accelerated rendering functions. For the most part the hardware drawing functions perform any kind of clipping, so it is up to the application to ensure that all coordinates passed to the accelerated rendering functions are within the range of the currently active drawing buffer. The exception to this rule is the ClipMonoImage, StretchBlt and clipped line drawing family of functions.

## Color Values

All color values passed to the accelerated rendering functions are packed pixel values, that will need to be pre-packed into the proper format required by the current framebuffer mode. In 8 bit color index modes, this is simply a color index between 0 and 255. In the 15 bits per pixel and above modes, you will need to pack the color values according to the RGB pixel format information stored in the *GA\_modeInfo* information block. The RGB pixel format information specifies the mask size and bit positions for all

red, green, blue and alpha components (alpha is generally ignored for framebuffer values). These mask should be used by the application to pack the appropriate RGB color values into a 32-bit integer to be passed to the appropriate rendering routines.

Currently the alpha component in 15-bit and 32-bits per pixel modes is unused, and should *always* be set to 0 for normal 2D rendering operations, as on some controllers these bits may be significant. For alpha blended operations, these bits can be used to control the blending of pixel colors during rendering.

## Direct Framebuffer Access

---

In order to allow both direct framebuffer access and hardware accelerator access, contention for video memory between the application program and the hardware accelerator must be properly handled. This is provided by the *EnableDirectAccess* and *DisableDirectAccess* functions. If the *EnableDirectAccess* function pointer is not NULL, then you *must* call these functions prior to performing any direct access to the framebuffer via either the banked framebuffer or the linear framebuffer, and then call *DisableDirectAccess* before calling any other hardware accelerator functions again.

If the *EnableDirectAccess* function is NULL, you *must* instead call the *WaitTillIdle* function to ensure the hardware has completed the last drawing operation before accessing the framebuffer directly. You need not call any other functions to return to accelerated rendering again.

When the video mode is first initialized, the hardware is automatically set up for hardware accelerated rendering, as though *DisableDirectAccess* was called immediately after setting the display mode.

## Hardware Triple Buffering

---

Hardware triple buffering is supported in the SciTech SNAP Graphics Architecture specification by allowing the application to schedule a display start address change, and then to later determine the status of the last scheduled display start address change. The *SetDisplayStart* function is used to schedule the display start address change, and the *GetDisplayStartStatus* function is used to determine the status of the last scheduled change. The following steps outline how to use hardware triple buffering:

1. Display from the first visible buffer, and render to the second hidden buffer.
2. Schedule a page flip for the second hidden buffer that you just finished rendering with a call to *SetDisplayStart* with the *waitVRT* flag set to 0, and start rendering immediately to the third hidden buffer. The CRT controller will be currently displaying from the first buffer.
3. Before scheduling the page flip for the third hidden buffer, wait until the last scheduled change has occurred by calling *GetDisplayStartStatus* until it returns non-zero. Schedule the page flip for the third hidden buffer (call *SetDisplayStart* with the *waitVRT* flag set to 0 again) and immediately begin rendering to the first

hidden buffer. The CRT controller will be currently displaying from the second buffer.

4. Repeat step 3 over and over cycling through each of the buffers.

Although the above method does require a spin loop polling on the *GetDisplayStartStatus*, in most cases when this function is called the page flip will already have occurred and the spin loop will time out immediately. The only time that this cannot occur is if the application is drawing at a frame rate in excess of the current hardware refresh rate (i.e.: in excess of 60-85 frames per second), and the resulting frame rate for the application will be pegged at the hardware refresh rate.

## Using the Buffer Manager

---

The SciTech SNAP Graphics Architecture supports drawing on and copying video memory data to and from offscreen video memory and the display screen. Although the core functionality is provided via the *SetDrawBuffer* function, the SciTech SNAP Graphics Architecture also supports a higher level interface to video memory management via the Buffer Manager. The Buffer Manager functions are provided through the *GA\_bufferFuncs* function group, and implement support for allocating, freeing, displaying and copying data to and from offscreen video memory buffers. The buffers can be allocated entirely in video memory, entirely in system memory or to fall back to system memory when video memory runs out. The buffers can also be allocated as fixed or moveable buffers in video memory, where moveable buffers can be moved around in video memory by the buffer manager to collect free video memory for future buffer allocations. Buffers can also be implemented as discardable buffers with a system memory buffer cache, such that non-critical buffers can be pushed out to system memory automatically when more video memory is needed.

Using the Buffer Manager is actually quite simple. The first thing to do is to allocate the primary display buffer along with a set number of 'flippable' buffers that can be made visible for page flipping and animation. This is done using the *InitBuffers* function, where you will pass in the total number of flippable buffers that you need. To display a flippable buffer and make it visible, you would pass a pointer to the buffer to the *FlipToBuffer* function (pointers to the flip buffers can be obtained using the *GetFlippableBuffer* function). By default after initializing the buffer manager, all drawing will go to the first buffer or 'primary' buffer in the flippable buffers list. To make another buffer active for drawing, the *SetActiveBuffer* function is used. After this function is called, all drawing using the functions in the *GA\_2DRenderFuncs* function group will go to the newly active buffer.

Apart from flippable buffers, it is also possible to allocate arbitrary sized buffers in offscreen video memory using the *AllocBuffer* function. You can allocate buffers with many different types of attributes with dimensions smaller or larger than the current display mode (some hardware does not support offscreen buffers larger than the display mode though). Once you have allocated the buffers, you can use them just like flippable

buffers in that you can draw on them and copy between them, you just can't make them directly visible via the *FlipToBuffer* function.

Once you have allocated a buffer (whether it is a flippable buffer or a regular offscreen buffer), if you wish to directly access the video memory in the buffer, you **must** use the *LockBuffer* and *UnlockBuffer* functions. Once you have called the *LockBuffer* function, the *Surface* pointer of the *GA\_buf* structure will point to a linear memory access that can be used to directly access the surface of the buffer. Prior to calling the *LockBuffer* function, the *Surface* pointer will be set to NULL (and will be reset to NULL after calling the *UnlockBuffer* function). The *LockBuffer* function also returns a physical memory address of the buffer in video memory, which can be used to program the video memory address into hardware DMA engines (note that not all drivers will support returning a physical memory address, and this will be 0 if not supported).

You can also directly copy data from any buffer to another buffer using the *BitBltBuf* and related functions. This allows you to copy data between any source buffer and the currently active buffer, with all features normally available in the hardware such as color key transparency, stretching, blending and more.

---

*Note: Another reason to use the buffer manager is that internally the SciTech SNAP Graphics Architecture drivers may utilize the buffer manager when it is enabled to speed up some software rendering functions through partial hardware acceleration. Until the application or shell driver enables the buffer manager, that functionality will not be enabled and those functions will run entirely in software.*

*Also note that if you are using the DirectX SNAP driver on Microsoft Windows, you **must** use the buffer manager functions to manage offscreen video memory. Directly calling on this driver is not supported.*

---

## Hardware Video Overlay Functions

---

Hardware video playback with multiple, independent overlay windows can be implemented on compatible devices using the hardware video playback functions. The functions provided by the *GA\_videoFuncs* function group provide a simple interface to hardware video playback. The application software does all of the decoding of video frames, and the graphics driver can be used to display the video data with hardware stretching and color space conversion. The steps to use hardware video playback are as follows:

1. Determine the hardware video playback capabilities for the graphics mode with the *GetVideoModeInfo* function and examining the *VideoWindows* member of the *GA\_modeInfo* structure. There is one structure in the *VideoWindows* list for each available hardware overlay window. Each video window may have different capabilities, so be sure to look for the one that supports what you need (generally the first one in the list is the most capable).
2. Allocate an offscreen video buffer of the correct dimensions and pixel format for the source video image you wish to display using the *AllocVideoBuffer* function.



This function might fail if there is not enough video memory available, or you request an input pixel format that is not supported.

3. Set the video window output rectangle on the screen with the *SetVideoOutput* function. This function also lets you specify what features are used for displaying the video, such as interpolation and color keying.
4. Set the hardware video color key, if applicable, using the *SetVideoColorKey* function.
5. Call *StartVideoFrame* to set up for decoding the next frame of data. Calling this function effectively makes a call to *LockBuffer* for the video buffer, such that the video memory is locked down for direct memory access.
6. Decode the frame of data into the buffer returned by *AllocVideoBuffer* above.
7. Call *EndVideoFrame* to complete the decoding of the video data and display it on the screen.
8. Repeat steps 4-6 until the video playback is complete.

## Stereoscopic Liquid Crystal Shutter Glasses

---

Stereoscopic liquid crystal (LC) shutter glasses are a cheap, easy solution for getting real 3D stereoscopic imaging out of a standard PC with any standard monitor. LC shutter glasses work by constantly blanking out video information for each eye in a sequential fashion, allowing the user to see the left image for a fraction of a second followed by the right image, followed by the left image again etc. In order to make LC shutter glasses work effectively on PC based graphics controllers, some mechanism for changing the displayed video information at every vertical retrace is necessary. New hardware is available that will do this automatically, and the graphics device driver defines the software interface necessary to allow applications to use these new hardware features.

The steps involved in enabling free running stereoscopic support are as follows:

1. Set the mode via *SetVideoMode* (using a high refresh rate if possible, such as 120Hz to 150Hz).
2. Enable free running stereoscopic mode by calling *EnableStereoMode* with a value of TRUE.
3. Draw the images for both the left and right eye images by using the *SetDrawBuffer* function to draw the left and right eye buffers (either side by side or above/below formats will work). Once both left and right eye images are rendered, the visible display buffer can be swapped as per normal using the *SetStereoDisplayStart* function. This function takes the byte offset of the left and right images as parameters, and the hardware will automatically flip between the left and right eye images at every vertical retrace.

4. To temporarily disable stereoscopic rendering, call the *SetStereoDisplayStart* function with both the left and right start addresses set to the same buffer. Although the hardware will still be in stereo mode, the screen image will not change every vertical retrace period.
5. Disable free running stereoscopic mode by calling *EnableStereoMode* with a value of FALSE when you are done with stereoscopic viewing.

### Refresh rates and stereoscopic imaging

When the hardware is running in free running stereoscopic mode and an image or 3D scene is being viewed through LC shutter glasses, the user will see the resulting image at half the original refresh rate through the shutter glasses. Hence a normally acceptable display running at 60Hz becomes a hard to view display running at 30Hz stereoscopic. For this reason when running in stereoscopic modes it is desirable to significantly increase the refresh rate of the graphics mode to values as high as 120Hz to 150Hz (depending on the monitors capabilities), which provides for 60Hz or 75Hz refresh per eye in stereoscopic modes.

The graphics device driver has full support for refresh rate control when setting a display mode via the *SetVideoMode* function. Stereoscopic applications can use this functionality in combination with the VESA GTF standard to increase the refresh rate of the stereoscopic application to acceptable levels (see the above section on refresh rate control for more information).

### Software driven display start address swapping

If the hardware does not support a free running stereoscopic display mode, a timer driven, software driven stereoscopic display can be supported on some platforms. See the *GA\_softStereoInit* function for information on how to set up software stereo support on supported platforms.

## Developing for Maximum Compatibility

---

This section contains information relating to developing application software with maximum compatibility in mind, without sacrificing performance or features. Although this document defines how the specification should work, there are many different flavors of hardware out in the field. It is very important that you design your application with the following special cases in mind so that your application will run on the widest variety of hardware possible.

One of the common mistakes that many developers make when they first start developing graphics code, is to assume the graphics card they have in their system is a representative sample of what exists in the field. Although the driver interface will be identical on another graphics card, the capabilities and attributes that card reports may be very different. This section deals with explaining the most common pitfalls that plague developers first starting to develop graphics code.

### **Support Both 15-bit and 16-bits Per Pixel Modes**

Many new games and applications have support for 15-bit and 16-bits per pixel high color modes. If you wish to support these modes, don't make the mistake of assuming that all devices will support 16-bit high color modes, or that all devices will support 15-bit high color modes. There are devices in the field that support only 15-bit modes, and there are also devices in the field that support only 16-bit modes. Hence it is of vital importance that your application code support both of these color depths to ensure maximum compatibility.

### **Support Both 24-bit and 32-bits per Pixel Modes**

If you are developing code to support 24-bit true color rendering, be prepared to find controllers in the field that have the true color modes supported as either 24-bits per pixel (3 bytes per pixel) or 32-bits per pixel (4 bytes per pixel). Generally the 32-bits per pixel modes are faster because the pixels can be written with a single CPU double word access, however 32-bits per pixel modes require more memory than the equivalent 24-bit modes. Hence be prepared to find controllers in the field that have only 24-bit modes, only 32-bit modes or both.

### **Do Not Assume Support for Double Scanned Modes**

If you are developing a game or application that wishes to support 320x200, 320x240 or 400x300 modes (in any color depth), be prepared for situations where these modes do not exist. To be able to initialize these modes on today's hardware requires support for double scanning, and there are some controllers in the field that do not support this. On these controllers these modes can never be supported, so your application or game must be able to deal with the situation if these modes do not exist.

In lieu of these modes not being available, the controller may provide support for 320x400, 320x480 and 400x600 modes which do not require double scanning. One neat solution is to support these modes by rendering your frames to a system memory buffer with a resolution 320x200, 320x240 or 400x300, and then do a copy to display memory with a 2x vertical stretch (just duplicate every scanline twice in software). The end result will look identical to a real 320x200, 320x240 or 400x300 mode, and you will only lose a small amount in overall performance.

# Graphics Device Driver Overview

---

This section contains the function and data structure references for the SciTech SNAP Graphics Architecture API. The following overview sections group related functions together, to help you find the specific functions of interest.

## Overview of Global Functions

---

This section contains an overview of all the global functions in the SciTech SNAP Graphics Architecture API. The global functions are called directly as regular C function calls within the application or shell driver, and are accessible to the application by linking with the `n_ga` library.

### Driver Loading and Initialization Functions

These global functions provide support for loading and initializing the device driver, for querying groups of function pointers from the driver by the application software.

<code>GA_enumerateDevices</code>	<code>GA_errorMsg</code>
<code>GA_getGlobalOptions</code>	<code>GA_getSNAPConfigPath</code>
<code>GA_loadDriver</code>	<code>GA_loadRef2d</code>
<code>GA_loadRegionMgr</code>	<code>GA_queryFunctions</code>
<code>GA_readGlobalOptions</code>	<code>GA_saveGlobalOptions</code>
<code>GA_saveOptions</code>	<code>GA_setActiveDevice</code>
<code>GA_setGlobalOptions</code>	<code>GA_status</code>
<code>GA_unloadDriver</code>	<code>GA_unloadRef2d</code>
<code>GA_unloadRegionMgr</code>	<code>REF2D_loadDriver</code>
<code>REF2D_queryFunctions</code>	<code>REF2D_unloadDriver</code>

### Display Mode Management Functions

These global functions provide support for managing the list of available display modes, CRTCT timings and refresh rates.

<code>GA_addMode</code>	<code>GA_addRefresh</code>
<code>GA_computeCRTCTimings</code>	<code>GA_delMode</code>
<code>GA_disableVBEMode</code>	<code>GA_enableVBEMode</code>
<code>GA_getCRTCTimings</code>	<code>GA_getMaxRefreshRate</code>
<code>GA_loadModeProfile</code>	<code>GA_restoreCRTCTimings</code>
<code>GA_saveCRTCTimings</code>	<code>GA_saveModeProfile</code>
<code>GA_setCRTCTimings</code>	<code>GA_setDefaultRefresh</code>
<code>GA_useDoubleScan</code>	

## Rectangle Arithmetic Functions

These global macros provide simplified support for managing rectangles and performing simple arithmetic on them. Note that all these functions are actually implemented as C style macros for performance.

GA_equalRect	GA_emptyRect
GA_disjointRect	GA_sectRect
GA_sectRectFast	GA_sectRectCoord
GA_sectRectFastCoord	GA_unionRect
GA_unionRectCoord	GA_offsetRect
GA_insetRect	GA_ptInRect

## Monitor Detection Functions

These global functions provide support for detecting, parsing and saving to disk information about the attached display monitors via the VESA DDC Plug and Play specification.

DDC_init	DDC_initExt
DDC_readEDID	DDC_writeEDID
EDID_parse	GA_detectPnPMonitor
GA_getParsedEDID	GA_saveMonitorInfo

## Monitor Database Functions

These global functions provide support for managing the SciTech SNAP Graphics Architecture monitor database.

MDBX_close	MDBX_first
MDBX_flush	MDBX_getErrCode
MDBX_getErrorMsg	MDBX_importINF
MDBX_insert	MDBX_last
MDBX_next	MDBX_open
MDBX_prev	MDBX_update

## Monitor Command Set Functions

These global functions provide support for interfacing with VESA DDC/CI compatible monitors using the VESA Monitor Command Set (MCS) interface.

MCS_begin	MCS_beginExt
MCS_enableControl	MCS_end
MCS_getCapabilitiesString	MCS_getControlMax
MCS_getControlValue	MCS_getControlValues
MCS_getSelfTestReport	MCS_getTimingReport
MCS_isControlSupported	MCS_resetControl
MCS_saveCurrentSettings	MCS_setControlValue
MCS_setControlValues	

## Overview of Queried Function Groups

---

This section contains an overview of all the function groups available via the SciTech SNAP Graphics Architecture API. Unlike the global functions, these functions are not called directly, but instead are called via groups of function pointers that are returned by the GA\_queryFunctions function. As such each function group is relative to the specific group of functions that the function pointer belongs in. In the detailed function reference section, these functions are actually documented in the data structures section, underneath the named data structure containing the function pointers themselves.

### Display Driver Initialization Functions

These functions are members of the GA\_initFuncs group. The functions form the main device driver functions that allow the application to enumerate the available display modes, set a display mode as well as access various house keeping functions.

AlignLinearBuffer	GetActiveHead
GetCertifyInfo	GetClosestPixelClock
GetConfigInfo	GetCRTCTimings
GetCurrentRefreshRate	GetCurrentVideoModeInfo
GetCustomVideoModeInfo	GetCustomVideoModeInfoExt
GetDisplayOutput	GetMonitorInfo
GetNumberOfHeads	GetOptions
GetUniqueFilename	GetVideoMode
GetVideoModeInfo	GetVideoModeInfoExt
PerformDisplaySwitch	PollForDisplaySwitch
SaveCRTCTimings	SaveRestoreState
SetActiveHead	SetCRTCTimings
SetCustomVideoMode	SetDisplayOutput
SetGlobalRefresh	SetModeProfile
SetMonitorInfo	SetOptions
SetRef2dPointer	SetSoftwareRenderFuncs
SetVideoMode	SwitchPhysicalResolution

## Device Driver Control Functions

These functions are members of the GA\_driverFuncs group. The functions form the main device driver functions that allow the application to change the display start address, program the color palette, program the gamma correction ramp and synchronize with the vertical refresh of the display.

EnableStereoMode	GetCurrentScanLine
GetDisplayStartStatus	GetGammaCorrectData
GetGammaCorrectDataExt	GetPaletteData
GetPaletteDataExt	GetVSyncWidth
IsVSync	SetBank
SetDisplayStart	SetDisplayStartXY
SetGammaCorrectData	SetGammaCorrectDataExt
SetPaletteData	SetPaletteDataExt
SetStereoDisplayStart	SetVSyncWidth
WaitVSync	

## 2D Rendering State Functions

These functions are members of the GA\_2DStateFuncs group. The functions provide support for changing the hardware 2D rendering state for the driver, such as downloading 8x8 monochrome and color patterns, setting the mix mode and line stipple pattern.

BuildTranslateVector	DisableDirectAccess
EnableDirectAccess	IsIdle
Set8x8ColorPattern	Set8x8MonoPattern
SetAlphaValue	SetBackColor
SetBlendFunc	SetDrawBuffer
SetForeColor	SetLineStipple
SetLineStippleCount	SetLineStyle
SetMix	SetPlaneMask
Use8x8ColorPattern	Use8x8MonoPattern
Use8x8TransColorPattern	Use8x8TransMonoPattern
WaitTillIdle	

## 2D Drawing Functions

These functions are members of the GA\_2DRenderFuncs group. These functions provide support for drawing all the different 2D primitives that the device driver supports.

BitBlt	BitBltBM
BitBltColorPatt	BitBltColorPattBM
BitBltColorPattLin	BitBltColorPattSys
BitBltFx	BitBltFxBM
BitBltFxLin	BitBltFxSys
BitBltFxTest	BitBltLin
BitBltPatt	BitBltPattBM
BitBltPattLin	BitBltPattSys
BitBltPlaneMasked	BitBltPlaneMaskedBM
BitBltPlaneMaskedLin	BitBltPlaneMaskedSys
BitBltSys	ClipEllipse
ClipMonoImageLSBBM	ClipMonoImageLSBLin
ClipMonoImageLSBSys	ClipMonoImageMSBBM
ClipMonoImageMSBLin	ClipMonoImageMSBSys
DrawBresenhamLine	DrawBresenhamStippleLine
DrawBresenhamStyleLine	DrawClippedBresenhamLine
DrawClippedBresenhamStippleLine	DrawClippedBresenhamStyleLine
DrawClippedLineInt	DrawClippedStippleLineInt
DrawClippedStyleLineInt	DrawColorPattEllipseList
DrawColorPattFatEllipseList	DrawColorPattRect
DrawColorPattScanList	DrawColorPattTrap
DrawEllipse	DrawEllipseList
DrawFatEllipseList	DrawLineInt
DrawPattEllipseList	DrawPattFatEllipseList
DrawPattRect	DrawPattScanList
DrawPattTrap	DrawRect
DrawRectExt	DrawRectLin
DrawScanList	DrawStippleLineInt
DrawStyleLineInt	DrawTrap
DstTransBlt	DstTransBltBM
DstTransBltLin	DstTransBltSys
GetBitmapBM	GetBitmapSys
GetPixel	PutMonoImageLSBBM
PutMonoImageLSBLin	PutMonoImageLSBSys
PutMonoImageMSBBM	PutMonoImageMSBLin
PutMonoImageMSBSys	PutPixel
SrcTransBlt	SrcTransBltBM
SrcTransBltLin	SrcTransBltSys
StretchBlt	StretchBltBM
StretchBltLin	StretchBltSys
UpdateScreen	



## Buffer Manager Functions

These functions are members of the GA\_bufferFuncs group. The functions provide support for allocating, freeing, copying and drawing on offscreen video memory buffers. The buffer manager functions should always be used to manage offscreen surfaces when possible, for maximum performance and compatibility.

AllocBuffer	BitBltBuf
BitBltColorPattBuf	BitBltFxBuf
BitBltPattBuf	BitBltPlaneMaskedBuf
DrawRectBuf	DstTransBltBuf
FlipToBuffer	FlipToStereoBuffer
FreeBuffer	GetClipper
GetFlippableBuffer	GetFlipStatus
GetPrimaryBuffer	InitBuffers
LockBuffer	SetActiveBuffer
SetClipper	SrcTransBltBuf
StretchBltBuf	UnlockBuffer
UpdateCache	UpdateFromCache
WaitTillFlipped	

## Complex Region Management Functions

These functions are members of the GA\_regionFuncs group. The functions provide support for allocating, freeing, copying and performing arithmetic on complex regions. Complex regions are used by graphical user interface environments to provide complex visible region clipping support.

ClearRegion	CopyIntoRegion
CopyRegion	DiffRegion
DiffRegionRect	FreeRegion
IsEmptyRegion	IsEqualRegion
NewRectRegion	NewRegion
OffsetRegion	OptimizeRegion
PtInRegion	SectRegion
SectRegionRect	TraverseRegion
UnionRegion	UnionRegionOfs
UnionRegionRect	GA_isSimpleRegion

## Hardware Video Overlay Functions

These functions are members of the GA\_videoFuncs group. The functions provide support for multiple hardware video overlay windows, allowing the application to define the input and output dimensions of the video overlay, as well as being able to start and end decoding for each video frame.

AllocVideoBuffer

FreeVideoBuffer

SetVideoOutput

EndVideoFrame

SetVideoColorKey

StartVideoFrame

## Hardware Cursor Functions

These functions are members of the GA\_cursorFuncs group. The functions provide support for hardware cursors, allowing the application to change the cursor shape, change its position and make it visible on the screen.

BeginAccess

IsHardwareCursor

SetColorCursor256

SetColorCursorRGBA

SetMonoCursor

ShowCursor

EndAccess

SetColorCursor

SetColorCursorRGB

SetCursorPos

SetMonoCursorColor



# *Graphics Device Driver Reference*

---

This section contains the function reference for the SciTech SNAP, Graphics Architecture.

## *External Functions*

---

## DDC\_init

Checks to see if DDC communication is available.

### Declaration

```
int NAPI DDC_init(  
    GA_devCtx *dc)
```

### Prototype In

snap/ddc.h

### Parameters

*dc*                SNAP device driver to use for communications

### Return Value

One of the *DDC\_errCode* error return values.

### Description

Obsolete. Use *DDC\_initExt()* instead.

### See Also

*DDC\_initExt*

## DDC\_initExt

Checks to see if DDC communication is available.

### Declaration

```
int NAPI DDC_initExt(
    GA_devCtx *dc,
    N_int32 iChannel)
```

### Prototype In

snap/ddc.h

### Parameters

<i>dc</i>	SNAP device driver to use for communications
<i>iChannel</i>	DDC channel to use for communications (0 for primary monitor)

### Return Value

One of the *DDC\_errCode* error return values.

### Description

This function initializes the DDC communications module. After checking that the I2C interface is working, this function will then attempt to initialize the DDC communication channel and verify that DDC2B communication is possible. If DDC2B is not available, this function will return a value of *ddcNoCommunication* (ie: cannot communicate with slave). The most likely cause of this failure condition is that there is no DDC capable monitor attached to the graphics device.

---

**Note:** *If this function returns a value of *ddcNotAvailable*, it means the installed graphics device does not support DDC communications.*

---

### See Also

*DDC\_readEDID*, *DDC\_initExt*

## DDC\_readEDID

Attempts to read the EDID information block from the monitor.

### Declaration

```
ibool NAPI DDC_readEDID(
    N_int32 slaveAddr,
    uchar *edid,
    N_int32 length,
    N_int32 blockNumber,
    N_int32 iChannel)
```

### Prototype In

snap/ddc.h

### Parameters

<i>slaveAddr</i>	Slave address to read the EDID data from
<i>edid</i>	Place to store the EDID information read
<i>length</i>	Number of bytes of EDID data to read
<i>blockNumber</i>	EDID block number to read (generally 0)
<i>iChannel</i>	DDC channel to use for communications (0 for primary monitor)

### Return Value

True on success, false for invalid checksum or communications error.

### Description

This function attempts to read the EDID information from the DDC2B slave. This function also does a checksum on the incoming EDID data, and if the checksum fails will return false. The slave address that is passed to this function should be 0xA0 to read the regular 128 byte EDID for the DDC 2.0 specification.

For DDC 3.0 you can pass in values of 0xA2 and 0xA6 for the Plug and Display and FPD 256 byte extended EDID blocks respectively.

### See Also

*DDC\_initExt*, *EDID\_parse*



## DDC\_writeEDID

Attempts to write the EDID information block to the monitor.

### Declaration

```
ibool NAPI DDC_writeEDID(
    GA_devCtx *dc,
    N_int32 slaveAddr,
    uchar *edid,
    N_int32 length,
    N_int32 blockNumber,
    N_int32 iChannel)
```

### Prototype In

snap/ddc.h

### Parameters

<i>dc</i>	SNAP device driver to use for communications
<i>slaveAddr</i>	Slave address to write the EDID data to (ie: 0xA0 for EDID 2)
<i>edid</i>	Place to store the EDID information read
<i>length</i>	Number of bytes of EDID data to read
<i>blockNumber</i>	EDID block number to write (generally 0)
<i>iChannel</i>	DDC channel to use for communications (0 for primary monitor)

### Return Value

True on success, false on error.

### Description

This function attempts to write the EDID information to the monitor. Most monitors allow writing of the EDID data to the monitor so that they can be configured during factory testing and setup. This feature is used to reprogram the EDID information in a monitor in the field. Note that in order to re-program the EDID information in the monitor, you need to ensure that the VSYNC is held high for the duration of writes to the EDID, as this is the write protection used by the monitor.

### See Also

*DDC\_initExt*, *DDC\_readEDID*

## EDID\_parse

Parse the binary EDID block from the monitor into useful information.

### Declaration

```
int NAPI EDID_parse(
    uchar *edid,
    EDID_record *rec,
    N_int32 requireDescriptor)
```

### Parameters

<i>edid</i>	EDID information block to parse
<i>rec</i>	Place to store monitor config record
<i>requireDescriptor</i>	True if the the descriptor block is required

### Return Value

1 if valid EDID, 2 if old EDID, 0 if not found.

### Description

This function parses the information in the EDID information block and attempts to fill in a structure containing detailed, useful information. Specifically this function finds information such as the maximum horizontal and vertical frequencies, maximum resolution and a list of all known standard and detailed timings. It also extracts the monitor manufacturer name, model name, serial numbers, manufacture date and other useful information.

If you pass a value of true for *requireDescriptor* and the EDID block does not contain valid operational limits information, this function will return a value of 2. This indicates that the EDID was parsed successfully, however the operational limits are missing from the data. If you call this function again with *requireDescriptor* set to false, the returned operational limits are derived from the list of standard and detailed timings in the EDID block, so may not be completely accurate for the attached monitor.

### See Also

*DDC\_initExt*, *DDC\_readEDID*

## GA\_addMode

Add a custom display mode to the device driver.

### Declaration

```
ibool NAPI GA_addMode(
    GA_devCtx *dc,
    N_int32 xRes,
    N_int32 yRes,
    N_int32 bits,
    N_int32 saveToDisk)
```

### Prototype In

snap/graphics.h

### Parameters

<i>dc</i>	Device context for driver to add mode to
<i>xRes</i>	X resolution of the mode to add
<i>yRes</i>	Y resolution of the mode to add
<i>bits</i>	Bits per pixel for the mode to add
<i>saveToDisk</i>	True if the mode profile should be saved to disk

### Return Value

True on success, or false if the mode could not be added.

### Description

This function is used to add a new display mode to a device. By default a standard table of display modes is available in the device, however this function allows the user to create custom display modes and add those to the SNAP drivers.

Unless you pass a value of true to the saveToDisk parameter, the changes will only affect the currently loaded copy of SNAP and will not affect new versions of SNAP that are loaded from disk. To make the change permanent, you need to set this parameter to true.

---

**Note:** *In order to enable support for the newly added mode for VESA VBE applications, you must also call the GA\_enableVBEMode for the same resolution and color depth before it will show up.*

---

### See Also

GA\_addRefresh, GA\_delMode, GA\_enableVBEMode

## GA\_addRefresh

Add a custom refresh rate for a mode to the CRTC database.

### Declaration

```
ibool NAPI GA_addRefresh(
    GA_devCtx *dc,
    N_int32 xRes,
    N_int32 yRes,
    N_int32 refresh,
    N_int32 saveToDisk)
```

### Prototype In

snap/graphics.h

### Parameters

<i>dc</i>	Device context for driver to add mode to
<i>xRes</i>	X resolution of the mode to add
<i>yRes</i>	Y resolution of the mode to add
<i>refresh</i>	New refresh rate to add
<i>saveToDisk</i>	True if the mode profile should be saved to disk

### Return Value

True on success, or false if the refresh rate could not be added.

### Description

This function is used to add a custom refresh rate to the CRTC database for a device. By default a standard database of known display modes is available, but this function allows the user to create custom refresh rates and add those to the SNAP drivers.

Unless you pass a value of true to the saveToDisk parameter, the changes will only affect the currently loaded copy of SNAP and will not affect new versions of SNAP that are loaded from disk. To make the change permanent, you need to set this parameter to true.

### See Also

*GA\_addMode*, *GA\_delMode*

## GA\_computeCRTCTimings

Compute the VESA GTF timings for a display mode

### Declaration

```
ibool NAPI GA_computeCRTCTimings (
    GA_devCtx *dc,
    GA_modeInfo *modeInfo,
    N_int32 refreshRate,
    N_int32 interlaced,
    GA_CRTCInfo *crtc,
    N_int32 pureGTF)
```

### Prototype In

snap/graphics.h

### Parameters

<i>dc</i>	Device context for driver to compute timings for
<i>modeInfo</i>	SuperVGA mode information block for the mode
<i>refreshRate</i>	Desired refresh rate to generate timings for
<i>interlaced</i>	True if the CRTC timings should be interlaced
<i>crtc</i>	Place to store the computed CRTC timings
<i>pureGTF</i>	True to use pure GTF formulas

### Return Value

True on success, or false if refresh rate is out of range.

### Description

This function computes a set of CRTC timings for the specific graphics mode at the specified refresh rate. The CRTC timings are computed using the VESA GTF timing formulas, combined with information returned by the SNAP driver. Note that the exact refresh rate that you get will not be exactly what you requested, and the exact value is returned in the CRTC information block block.

If the pureGTF parameter is set to true, we will leave the CRTC timings as is after the GTF formulas have computed the appropriate timings for the refresh rate given the closest available pixel clock. This means that we may not be able to hit a specific refresh rate exactly, but the CRTC timings will be GTF compliant. In some cases it may be more beneficial to hit an exact refresh rate on the nail (such as when viewing stereo images) and in order to do this we will modify the horizontal and vertical totals slightly to compensate for the granularity in the pixel clock (which in some cases may be quite large).

### See Also

*GA\_getMaxRefreshRate*

## GA\_delMode

Delete a display mode for the device.

### Declaration

```
ibool NAPI GA_delMode(
    GA_devCtx *dc,
    N_int32 xRes,
    N_int32 yRes,
    N_int32 bits,
    N_int32 saveToDisk)
```

### Prototype In

snap/graphics.h

### Parameters

<i>dc</i>	Device context for driver to add mode to
<i>xRes</i>	X resolution of the mode to add
<i>yRes</i>	Y resolution of the mode to add
<i>bits</i>	Bits per pixel for the mode to add
<i>saveToDisk</i>	True if the mode profile should be saved to disk

### Return Value

True on success, or false if the mode could not be deleted.

### Description

This function is used to add a delete a display mode for a device. By default a standard table of display modes is available in the device, however this function allows the user to remove unwanted display modes from the SNAP drivers.

Unless you pass a value of true to the saveToDisk parameter, the changes will only affect the currently loaded copy of SNAP and will not affect new versions of SNAP that are loaded from disk. To make the change permanent, you need to set this parameter to true.

### See Also

*GA\_addMode*, *GA\_addRefresh*, *GA\_disableVBEMode*

## GA\_detectPnPMonitor

Detects a Plug and Play monitor attached to the graphics card

### Declaration

```
int NAPI GA_detectPnPMonitor(
    GA_devCtx *dc,
    GA_monitor *monitor,
    ibool *hasChanged)
```

### Prototype In

snap/graphics.h

### Parameters

<i>dc</i>	Device context for the device to do detection for
<i>monitor</i>	Place to store the returned monitor information.
<i>hasChanged</i>	Place to store whether the monitor has changed

### Return Value

0 if no DDC monitor, 1 if found, 2 if found without operational limits

### Description

This function performs DDC monitor detection for the passed in device context. If the monitor is found, it is returned in the monitor parameter. Note that if we detect an older DDC monitor that does not have operational limits, we use the PNPID from the monitor EDID to search for the correct entry in our monitor database and return those values instead. This allows us to correct for badly formed monitor EDID information present in early monitors (ie: EDID's prior to 1.1).

Note that this function will also return whether the monitor information has changed from the monitor record passed into the function. If the values have changed in anyway the 'changed' parameter will be set to true otherwise it will be set to false.

---

**Note:** *This function assumes that the system is already in graphics mode. Some device's will not allow PnP monitor detection unless the graphics card is running in hires graphics mode, so you should ensure this is the case before calling this function.*

---

### See Also

GA\_saveMonitor, GA\_getParsedEDID

## GA\_disableVBEMode

Disable a display mode for use by VESA VBE applications.

### Declaration

```
ibool NAPI GA_disableVBEMode(
    GA_devCtx *dc,
    N_int32 xRes,
    N_int32 yRes,
    N_int32 bits,
    N_int32 saveToDisk)
```

### Prototype In

snap/graphics.h

### Parameters

<i>dc</i>	Device context for driver to add mode to
<i>xRes</i>	X resolution of the mode to add
<i>yRes</i>	Y resolution of the mode to add
<i>bits</i>	Bits per pixel for the mode to add
<i>saveToDisk</i>	True if the mode profile should be saved to disk

### Return Value

True on success, or false if the mode could not be deleted.

### Description

This function is used to disable a specific display mode for use by VESA VBE applications in a DOS box environment.

Unless you pass a value of true to the *saveToDisk* parameter, the changes will only affect the currently loaded copy of SNAP and will not affect new versions of SNAP that are loaded from disk. To make the change permanent, you need to set this parameter to true.

### See Also

*GA\_enableVBEMode*, *GA\_addMode*, *GA\_delMode*



## GA\_disjointRect

Determines if two rectangles are disjoint.

### Declaration

```
ibool GA_disjointRect(  
    GA_rect r1,  
    GA_rect r2)
```

### Prototype In

snap/graphics.h

### Parameters

<i>r1</i>	First rectangle to test
<i>r2</i>	Second rectangle to test

### Return Value

True if the rectangles are disjoint, false if they overlap.

### Description

This function determines whether two rectangles are disjoint, which is true if the rectangles do not overlap at any coordinates.

### See Also

*GA\_emptyRect*, *GA\_equalRect*, *GA\_unionRect*, *GA\_sectRect*

## GA\_emptyRect

Determines if a rectangle is empty.

### Declaration

```
ibool GA_emptyRect(  
    GA_rect r)
```

### Prototype In

snap/graphics.h

### Parameters

*r*            The rectangle to test

### Return Value

True if the rectangle is empty, otherwise false.

### Description

Determines if a rectangle is empty or not. A rectangle is defined as being empty if the right coordinate is less than or equal to the left coordinate, or if the bottom coordinate is less than or equal to the top coordinate.

## GA\_enableVBEMode

Enable a display mode for use by VESA VBE applications.

### Declaration

```
ibool NAPI GA_enableVBEMode(
    GA_devCtx *dc,
    N_int32 xRes,
    N_int32 yRes,
    N_int32 bits,
    N_int32 saveToDisk)
```

### Prototype In

snap/graphics.h

### Parameters

<i>dc</i>	Device context for driver to add mode to
<i>xRes</i>	X resolution of the mode to add
<i>yRes</i>	Y resolution of the mode to add
<i>bits</i>	Bits per pixel for the mode to add
<i>saveToDisk</i>	True if the mode profile should be saved to disk

### Return Value

True on success, or false if the mode could not be added.

### Description

This function is used to enable a specific display mode for use by VESA VBE applications in a DOS box environment. Many early VESA applications do not correctly handle large mode lists returned by the VBE driver, so only a specific subset of modes are actually reported to the VBE application by the SNAP VBE emulation driver. This function is used to enable support for specific modes that the user desires on a mode by mode basis.

Unless you pass a value of true to the saveToDisk parameter, the changes will only affect the currently loaded copy of SNAP and will not affect new versions of SNAP that are loaded from disk. To make the change permanent, you need to set this parameter to true.

---

**Note:** *Modes added to this list only affect the filtering mechanism for VESA VBE applications. A mode will not be available unless it is first added to the standard mode list for the SNAP driver using GA\_addMode (or if the mode is a standard mode already supported).*

---

### See Also

GA\_disableVBEMode, GA\_addMode, GA\_delMode

## GA\_enumerateDevices

Enumerates the number of display devices on the PCI bus.

### Declaration

```
int NAPI GA_enumerateDevices(  
    N_int32 shared)
```

### Prototype In

snap/graphics.h

### Parameters

*shared*                      True if the device driver should be loaded into shared memory

### Return Value

Number of display devices found.

### Description

This function enumerates the number of available display devices on the PCI bus, and returns the number found. The number of devices allows you to load graphics drivers for each device, but calling *GA\_loadDriver* starting with device 0 and up to the maximum number of devices-1. Note that even though we may enumerate a valid PCI display device, it is possible that no graphics driver is available for the device and the *GA\_loadDriver* function could fail.

Hence this function does not enumerate all the devices and match them up with valid graphics drivers; the matching to available drivers is done when you call *GA\_loadDriver*.

### See Also

*GA\_loadDriver*

## GA\_equalRect

Compares two rectangles for equality.

### Declaration

```
ibool GA_equalRect(  
    GA_rect r1,  
    GA_rect r2)
```

### Prototype In

snap/graphics.h

### Parameters

<i>r1</i>	First rectangle to compare
<i>r2</i>	Second rectangle to compare

### Return Value

True if the rectangles are equal, false if not.

### Description

Compares two rectangles for equality.

## GA\_errorMsg

Returns a descriptive string for a device loader error code.

### Declaration

```
const char * NAPI GA_errorMsg(  
    N_int32 status)
```

### Prototype In

snap/graphics.h

### Parameters

*status*                      Status code to get string for

### Return Value

String describing error condition.

### Description

Returns a descriptive string for a device loader error code.

---

**Note:** *The string is returned in a temporary buffer that is reused. Hence the calling application should copy the returned string to a different buffer immediately after calling this function and before calling any other SNAP functions.*

---

### See Also

GA\_status

## GA\_getCRTCTimings

Get the CRTC timings for a specific display mode and refresh rate

### Declaration

```
ibool NAPI GA_getCRTCTimings(
    GA_devCtx *dc,
    GA_modeInfo *modeInfo,
    N_int32 refreshRate,
    GA_CRTCInfo *crtc)
```

### Prototype In

snap/graphics.h

### Parameters

<i>dc</i>	Device context to save CRTC timings information for
<i>modeInfo</i>	Display mode information block
<i>refreshRate</i>	Refresh rate to retrieve the CRTC timings for
<i>crtc</i>	Place to store the retrieved CRTC timings

### Return Value

True on success, false on error.

### Description

This function obtains the CRTC timings for a particular display mode and refresh rate from the internal CRTC timings database. If the resolution and/or refresh rate is not supported by the display mode, this function returns false.

### See Also

*GA\_saveCRTCTimings*, *GA\_restoreCRTCTimings*, *SetCRTCTimings*

## GA\_getCurrentRef2d

Gets a copy of the current SNAP reference rasteriser for the specified device index.

### Declaration

```
REF2D_driver * NAPI GA_getCurrentRef2d(
    N_int32 deviceIndex)
```

### Prototype In

snap/graphics.h

### Parameters

<i>deviceIndex</i>	Index of the device to obtain the SNAP reference rasteriser for
--------------------	---

### Return Value

Pointer to current SNAP reference rasteriser on success, NULL on failure.

### Description

This returns a copy of the current SNAP reference rasteriser driver for the specified device index. This function will return NULL if the driver has not been loaded (ie: SciTech Display Doctor is not the primary display driver in the system). The primary purpose of this function is to allow operating system utilities and applications to access a copy of the existing, shared SNAP driver used by the display driver in the system.

---

**Note:** *Do not call GA\_unloadRef2d with the value returned from this function!!*

---

### See Also

GA\_loadRef2d, GA\_getCurrentDriver



## GA\_getDaysLeft

Returns the number of days left in the evaluation license

### Declaration

```
int NAPI GA_getDaysLeft(  
    N_int32 shared)
```

### Prototype In

snap/graphics.h

### Return Value

Number of days left in evaluation license, -1 if registered, 0 if timed out.

### Description

This function checks the number of days left in the evaluation period. If the product has been registered this function returns a value of -1, otherwise it returns the number of days left in the evaluation period.

## GA\_getDisplaySerialNo

Returns the user serial number as an ASCII string

### Declaration

```
const char * NAPI GA_getDisplaySerialNo(ibool shared)
```

### Prototype In

snap/graphics.h

### Parameters

*shared*                      True if the device driver should be loaded into shared memory

### Return Value

Serial number as an ASCII string.

### Description

This function can be used to get an ASCII string version of the user serial number that can be displayed for the user. This is a printable version of the serial number, and does not contain the actual registration code information (ie: it is a serial number that can be used to look up the user in a customer database).

OEM versions of the product always return "N/A" for this function.

---

**Note:** *The string is returned in a temporary buffer that is reused. Hence the calling application should copy the returned string to a different buffer immediately after calling this function and before calling any other SNAP functions.*

---

## GA\_getDisplayUserName

Returns the user name as an ASCII string

### Declaration

```
const char * NAPI GA_getDisplayUserName(ibool shared)
```

### Prototype In

snap/graphics.h

### Parameters

*shared*                      True if the device driver should be loaded into shared memory

### Return Value

User name or OEM company name as an ASCII string.

### Description

This function can be used to get an ASCII string version of the user name or OEM company name (for OEM versions) that that can be displayed for the user.

---

**Note:** *The string is returned in a temporary buffer that is reused. Hence the calling application should copy the returned string to a different buffer immediately after calling this function and before calling any other SNAP functions.*

---

## GA\_getFakePCIID

Returns the internal fake PCI ID for ISA/VLB devices

### Declaration

```
const char * NAPI GA_getFakePCIID(void)
```

### Prototype In

snap/graphics.h

### Description

This internal function returns the fake PCI device ID as a string from the library. This value is only valid for ISA/VLB devices, and is used so that ISA/VLB devices can be cleanly integrated into the PCI detection and certification system.

---

**Note:** *The string is returned in a temporary buffer that is reused. Hence the calling application should copy the returned string to a different buffer immediately after calling this function and before calling any other SNAP functions.*

---

## GA\_getGlobalOptions

Returns the current global options from the graphics device driver.

### Declaration

```
void NAPI GA_getGlobalOptions(
    GA_globalOptions *options,
    ibool shared)
```

### Prototype In

snap/graphics.h

### Parameters

<i>options</i>	Place to store the returned options information
<i>shared</i>	True if the device driver should be loaded into shared memory

### Description

This function returns a structure which contains global configuration options effective for the installed devices.

---

**Note:** *The dwSize member of the profile structure is intended for future compatibility, and must be set to the size of the structure before calling this function. Only the number of bytes set in the dwSize member will be copied into the callers structure.*

---

### See Also

GA\_setGlobalOptions, GA\_saveGlobalOptions

## GA\_getInternalName

Returns the internal binary driver name for the device

### Declaration

```
const char * NAPI GA_getInternalName(  
    N_int32 deviceIndex)
```

### Prototype In

snap/graphics.h

### Parameters

*deviceIndex*                      Index of device to get driver name for

### Description

This internal function returns the internal binary driver name for the device. This information is only useful for QA purposes, and may change over time.

---

**Note:** *The string is returned in a temporary buffer that is reused. Hence the calling application should copy the returned string to a different buffer immediately after calling this function and before calling any other SNAP functions.*

---

## GA\_getLicensedDevices

Returns the list of licensed device types for the license files

### Declaration

```
N_uint32 * NAPI GA_getLicensedDevices (
    ibool shared)
```

### Prototype In

snap/graphics.h

### Parameters

*shared*                      True if the device driver should be loaded into shared memory

### Return Value

List of licensed device types, NULL if not an OEM licensed product.

### Description

This function can be used to get a numerical list of all licensed device types stored in the license file. This is used by control panel applets like the DDStereo control panel to determine what stereo glasses types are licensed by the OEM vendor. This list is always terminated with a value of 0x10101010.

---

**Note:** *The array returned by this function lives in a temporary buffer that is reused. Hence the calling application should copy the returned array to a different buffer immediately after calling this function and before calling any other SNAP functions.*

---

## GA\_getMaxRefreshRate

Determine the maximum refresh rate for a particular display mode

### Declaration

```
void NAPI GA_getMaxRefreshRate(
    GA_devCtx *dc,
    GA_modeInfo *modeInfo,
    N_int32 interlaced,
    float *maxRefresh)
```

### Prototype In

snap/graphics.h

### Parameters

<i>dc</i>	SNAP device context to use
<i>modeInfo</i>	SuperVGA mode information block for the mode
<i>interlaced</i>	True if the mode should be interlaced or not
<i>maxRefresh</i>	Place to store maximum refresh rate in Hz.

### Description

This function computes the maximum refresh rate for the graphics modes specified by the passed in mode information block. The maximum refresh rate is computed from the information returned by the SNAP driver, using CRTC values computed using the VESA GTF formulas.

### See Also

*GA\_computeCRTCTimings*



## GA\_getParsedEDID

Returns the parsed EDID record if a DDC monitor was found

### Declaration

```
int NAPI GA_getParsedEDID(  
    GA_devCtx *dc,  
    EDID_record *rec)
```

### Prototype In

snap/graphics.h

### Parameters

<i>dc</i>	Device context to retrieve the EDID record for
<i>rec</i>	Place to return the parsed edid record

### Return Value

True if DDC monitor found and EDID was valid, false if no valid DDC monitor

### Description

This function returns the existing, parsed EDID record for the primary monitor for the specific device context. If no valid EDID record was obtained from the monitor when the driver was loaded, this function returns false.

### See Also

*GA\_detectPnPMonitor*

## GA\_getSNAPConfigPath

Returns the currently active SNAP configuration file path

### Declaration

```
const char * NAPI GA_getSNAPConfigPath(void)
```

### Prototype In

snap/graphics.h

### Return Value

Path to currently active SNAP configuration files

### Description

This function returns the path to the currently active SNAP configuration files on the system. User applications that need to access the monitor database and other configuration files directly should use this function to find the location of those files, rather than the PM\_getSNAPConfigPath function. The reason is that the PM library version always returns the current system global configuration path, while this versions returns the currently active versions. It is possible for SNAP to be located relative to the application directory, and this function will return the correct path if this is the case.

---

**Note:** *The string is returned in a temporary buffer that is reused. Hence the calling application should copy the returned string to a different buffer immediately after calling this function and before calling any other SNAP functions.*

---

## GA\_insetRect

Insets the coordinates of a rectangle.

### Declaration

```
void GA_insetRect(  
    GA_rect r,  
    int dx,  
    int dy)
```

### Prototype In

snap/graphics.h

### Parameters

<i>r</i>	Rectangle to inset
<i>dx</i>	Amount to inset the x coordinates by
<i>dy</i>	Amount to inset the y coordinates by

### Description

This functions insets the rectangle by the specified values. This function effectively performs the following operation on the rectangle:

```
left += dx;  
top += dy;  
right -= dx;  
bottom -= dy;
```

### See Also

*GA\_offsetRect*

## GA\_isLiteVersion

Returns true if the product is the Lite retail product.

### Declaration

```
int NAPI GA_isLiteVersion(ibool shared)
```

### Prototype In

snap/graphics.h

### Parameters

*shared*            True if the device driver should be loaded into shared memory

### Return Value

True if the Lite retail product, false if not.

### Description

This function can be used to determine if the product is the Lite version of the retail product, or the full Profesional version. OEM versions of the product always return false for this function.

## GA\_isOEMVersion

Returns true if the product is an OEM licensed product.

### Declaration

```
int NAPI GA_isOEMVersion(  
    ibool shared)
```

### Prototype In

snap/graphics.h

### Parameters

*shared*                      True if the device driver should be loaded into shared memory

### Return Value

True if an OEM licensed product, false if not.

### Description

This function can be used to determine if the product is an OEM licensed version of SNAP, or if it is a retail end user version of the product. Retail end user versions always require a registration code.

## GA\_isSharedDriverLoaded

### Declaration

```
ibool          NAPI GA_isSharedDriverLoaded(void)
```

## GA\_isSimpleRegion

Returns true if a region is a simple region, otherwise false.

### Declaration

```
ibool GA_isSimpleRegion(  
    const GA_region *r)
```

### Prototype In

snap/graphics.h

### Parameters

*r*                Region to test.

### Description

This function determines if the region is simple or not. A simple region is one that consists of only a single rectangle. This function will not work properly if the region has been through a number of region algebra routines with other non-simple regions, even though the end result may be a single rectangle.

### See Also

*UnionRegion, DiffRegion, SectRegion*

## GA\_loadDriver

Loads a graphics driver file from disk for the specified device.

### Declaration

```
GA_devCtx * NAPI GA_loadDriver(
    N_int32 deviceIndex,
    N_int32 shared)
```

### Prototype In

snap/graphics.h

### Parameters

<i>deviceIndex</i>	Index of device to load driver for (0 for primary controller)
<i>shared</i>	True if the device driver should be loaded into shared memory

### Return Value

Pointer to the loaded graphics driver, or NULL on error.

### Description

Loads the driver file from disk and initialises the device context ready for use. If the driver file cannot be found, or the driver does not detect the installed hardware, we return NULL and the application can get the status code with the N\_status function.

Note that for multiple controller support you can specify the device you want to load the graphics driver for with the deviceIndex parameter. This parameter is 0 for the primary controller, 1 for the secondary controller and so on. The ordering of the controllers is determined by their enumeration order on the PCI bus, however device 0 is always the primary controller.

### See Also

*GA\_setActiveDevice, GA\_unloadDriver, N\_status, N\_errorMsg, GA\_isSharedDriverLoaded*



## GA\_loadInGUI

Force the amount of display memory configured for the driver.

### Declaration

```
ibool NAPI GA_loadInGUI(  
    N_int32 shared)
```

### Prototype In

snap/graphics.h

### Parameters

*shared*                      True if the device driver should be loaded into shared memory

### Return Value

True on success, false on error.

### Description

This function loads the SNAP driver and forces the memory size for subsequently loaded drivers to the specified size in Kb. The method to detect the amount of installed display memory on a graphics card in many cases requires the state of the hardware to be changed temporarily so that the memory size can be detected. This will cause corruption if a driver is being loaded when native IHV drivers are already running (ie: as a DDC control panel utility, or a stereo game driver utility). This function is used to force the memory size and avoid the hardware state change so that loading a driver is completely non-destructive to the hardware state. The memory size should be obtained using the native OS system services (such as DirectX).

### See Also

*GA\_loadDriver*

## GA\_loadModeProfile

Load the mode profile information from disk for the specified driver

### Declaration

```
void NAPI GA_loadModeProfile(
    GA_devCtx *dc,
    GA_modeProfile *profile)
```

### Prototype In

snap/graphics.h

### Parameters

<i>dc</i>	Device context to load mode profile information for
<i>profile</i>	Mode profile information block to load

### Description

This function loads the mode profile information from disk. The mode profile is the list of modes that the driver will report to user applications. You can modify this list to customise the set of display modes that the driver supports.

---

**Note:** *This function does not update the mode profile in the driver itself, but rather you should call SetModeProfile to update the internal mode profile first before saving the options to disk.*

---

### See Also

GA\_saveModeProfile

## GA\_loadRef2d

Loads the 2D reference rasteriser device driver chain, including filter drivers.

### Declaration

```
ibool NAPI GA_loadRef2d(
    GA_devCtx_FAR_ *dc,
    N_int32 shared,
    GA_modeInfo_FAR_ *modeInfo,
    N_int32 transferStart,
    REF2D_driver_FAR_ *_FAR_ *drv)
```

### Prototype In

snap/graphics.h

### Parameters

<i>dc</i>	Device to load reference rasteriser for
<i>shared</i>	True if the device driver should be loaded into shared memory
<i>modeInfo</i>	Mode information for the hardware display mode in use
<i>transferStart</i>	Start of the offscreen transfer buffer area in bytes
<i>drv</i>	Pointer to the place to store loaded driver address

### Return Value

True on success, false on failure.

### Description

This function loads a copy of the SNAP 2D Reference Rasteriser from disk for use with the specified hardware display mode. If the user has enabled rotation display support, or multi-controller display support, this function will load and initialise all the necessary SNAP filter drivers to enable those functions for the specified display mode.

The transferStart parameter is used to determine where the start of the offscreen display memory transfer buffer is located. This transfer buffer is used by the Portrait and Multi-Controller filter drivers for storing temporary bitmap images in offscreen display memory for enhanced performance. If the amount of memory from the start of the transfer area to the end of available offscreen display memory is large enough to hold the transfer buffer (no more than an entire display screen is ever required), then the filter drivers will use that memory as necessary. If the amount of available memory is less than required, the filter drivers will use system memory buffers instead. To disable the use of the transfer buffer, simply pass in a value that is equal to the amount of video memory on the graphics adapter in bytes.

### See Also

*GA\_unloadRef2d*, *GA\_unloadDriver*, *GA\_getCurrentRef2d*

## GA\_loadRegionMgr

Loads the 2D region manager driver.

### Declaration

```
ibool NAPI GA_loadRegionMgr(
    GA_regionFuncs *funcs,
    PE_MODULE **hModule,
    ulong *size)
```

### Prototype In

snap/graphics.h

### Parameters

<i>funcs</i>	Pointer to function structure to fill out
<i>hModule</i>	Place to store the module handle for the driver
<i>size</i>	Place to store the size of the loaded module

### Return Value

True on success, false on failure.

### Description

This function loads a new copy of the SNAP 2D region manager functions and exports the function list from the driver. Usually for SNAP graphics applications, the region functions can simply be obtained from the SNAP 2D reference rasteriser which also loads a global set of region functions for internal use. However SNAP applications can use this function to load a separate copy that is independent of the reference rasteriser for internal use by the application.

### See Also

*GA\_unloadRegionMgr*

## GA\_offsetRect

Offsets a rectangle by the specified amount.

### Declaration

```
void GA_offsetRect(  
    GA_rect r,  
    int dx,  
    int dy)
```

### Prototype In

snap/graphics.h

### Parameters

<i>r</i>	Rectangle to offset
<i>dx</i>	Amount to offset x coordinates by
<i>dy</i>	Amount to offset y coordinates by

### Description

This function offsets the specified rectangle by the dx and dy coordinates. This function effectively performs the following operation on the rectangle:

```
left += dx;  
top += dy;  
right += dx;  
bottom += dy;
```

### See Also

*GA\_insetRect*

## GA\_programMTRRegisters

Programs the MTR registers for the specific graphics device context

### Declaration

```
void NAPI GA_programMTRRegisters(  
    GA_devCtx *dc)
```

### Prototype In

snap/graphics.h

### Parameters

*dc*                      Device context to program MTR registers for

### Description

This is an internal function to program the MTR registers that is not usually called from outside application code. However this function is exposed so that it can be called from code that resets the machine to the correct state after coming out of a power management suspend mode.

## GA\_ptInRect

Returns true if supplied point is within the definition of a rectangle, otherwise false.

### Declaration

```
ibool GA_ptInRect(  
    int x,  
    int y,  
    GA_rect r)
```

### Prototype In

snap/graphics.h

### Parameters

<i>x</i>	X coordinate of rectangle to test
<i>y</i>	Y coordinate of rectangle to test
<i>r</i>	Rectangle to test

### Return Value

True if supplied point is within the definition of the specified rectangle.

### Description

This function tests whether a point is within the bounds of a rectangle or not.

## GA\_queryFunctions

Returns the function pointers for the specified function group.

### Declaration

```
ibool NAPI GA_queryFunctions(
    GA_devCtx *dc,
    N_uint32 id,
    void _FAR_ *funcs)
```

### Prototype In

snap/graphics.h

### Parameters

<i>dc</i>	Device driver to return the functions for
<i>id</i>	Identifier for the function group to get pointers for
<i>funcs</i>	Pointer to function block to fill in

### Return Value

True if the requested function group is available, false if not.

### Description

This function is the main function that is used by the graphics application code to get a block of function pointers for a specified function group. The function groups are defined *GA\_funcGroupsType* enumeration, and breaks up the functions in the device driver API into groups of logically similar functions. For instance to get the block of functions for the hardware cursor, you would call this function with the *GA\_GET\_CURSORFUNCS* identifier.

All application and device driver code must call this function and *never* call the *GA\_devCtx->loader.QueryFunctions* member directly. This is important because in many cases special procedures must be followed to allow ring 3 application code to directly call the function pointers returned by the internal function. The *GA\_devCtx* version always returns the actual device driver functions which may require that they be called at ring 0 on some platforms. This function insulates the developer from worrying about this as the functions returned by this function are always safe to be called from the context that the code is compiled (ie: this function should be used for all ring 0 device driver and ring 3 application level code).

---

**Note:** *To allow for future compatibility, all function blocks begin with a *dwSize* member. The caller is expected to fill in the *dwSize* member with the size of the function block being retrieved before calling *GA\_queryFunctions*. If the driver exports more functions than the application knows about, only a subset of the functions are copied to the application. If the application expects more functions than the driver provides, the non-existent functions are set to NULL pointers by *GA\_queryFunctions*, and the remainder copies from the driver.*

---



---

**Note:** *This mechanism also provides for a clean and simple upgrade path for future drivers, while ensuring maximum compatibility with existing specifications. New functions for a particular group can simply be added to the end of the function group to extend that group. Totally new function groups can be added by defining new identifiers for that function group, and older drivers will return a NULL if that function group is requested. Finally if a function group requires a complete redesign to achieve maximum performance for next generation hardware, a new extended function group can be defined (and drivers can continue to export the older and slower function group for backwards compatibility).*

---

**See Also**

*REF2D\_queryFunctions*

## GA\_readGlobalOptions

Reads the global options file directly from disk.

### Declaration

```
ibool NAPI GA_readGlobalOptions(  
    GA_globalOptions *options)
```

### Prototype In

snap/graphics.h

### Parameters

*options*                      Place to store the returned options information

### Return Value

False if no options file found.

### Description

This function is similar to *GA\_getGlobalOptions*, but it does not require SciTech SNAP Graphics to be loaded at all in order to read the global options file. This is useful for situations where the memory footprint of loading SciTech SNAP Graphics needs to be avoided.

---

**Note:** *The dwSize member of the profile structure is intended for future compatibility, and must be set to the size of the structure before calling this function. Only the number of bytes set in the dwSize member will be copied into the callers structure.*

---

### See Also

*GA\_getGlobalOptions, GA\_setGlobalOptions, GA\_saveGlobalOptions*

## GA\_registerLicense

Registers a linkable library license with the library.

### Declaration

```
int NAPI GA_registerLicense(  
    uchar *license,  
    N_int32 shared)
```

### Prototype In

snap/graphics.h

### Parameters

<i>license</i>	Pointer to the binary license to register
<i>shared</i>	True if the device driver should be loaded into shared memory

### Return Value

True on success, false on failure.

### Description

This function is used to register a linkable library license with the graphics library. This is provided for software vendors who have purchased a linkable library license to use and distribute the graphics device support library with their custom application software. Simply pass in a pointer to the binary license structure that was provided to you by a SciTech Software sales representative.

## GA\_restoreCRTCTimings

Restore the CRTC timings from disk for the specified driver

### Declaration

```
ibool NAPI GA_restoreCRTCTimings(  
    GA_devCtx *dc)
```

### Prototype In

snap/graphics.h

### Parameters

*dc*                      Device context to save CRTC timings information for

### Return Value

True on success, false on error.

### Description

This function restores the driver CRTC timings information by reading the current values from disk into the version the driver has loaded in memory. This is used to restore the timings to those present the last time the timings were saved to disk.

### See Also

*GA\_saveCRTCTimings*, *GetCRTCTimings*, *SaveCRTCTimings*, *SetGlobalRefresh*, *GA\_saveModeProfile*, *GA\_saveOptions*

## GA\_saveCRTCTimings

Save the CRTC timings to disk for the specified driver

### Declaration

```
ibool NAPI GA_saveCRTCTimings(  
    GA_devCtx *dc)
```

### Prototype In

snap/graphics.h

### Parameters

*dc*                      Device context to save CRTC timings information for

### Return Value

True on success, false on error.

### Description

This function saves the driver CRTC timings information to disk. If any of the timings for the driver have been changed via the SetCRTCTimings or SetGlobalRefresh functions, this will save those timings to disk to be used as the defaults from then on.

### See Also

*GA\_restoreCRTCTimings*, *GetCRTCTimings*, *SaveCRTCTimings*, *SetGlobalRefresh*, *GA\_saveModeProfile*, *GA\_saveOptions*

## GA\_saveGlobalOptions

Save the global options to disk

### Declaration

```
ibool NAPI GA_saveGlobalOptions(  
    GA_globalOptions *options)
```

### Prototype In

snap/graphics.h

### Parameters

*options*                      Global options information block to save

### Return Value

True on success, false on error (see *GA\_status* for error info)

### Description

This function saves the global options information to disk.

### See Also

*GA\_getGlobalOptions*, *GA\_setGlobalOptions*

## GA\_saveModeProfile

Save the mode profile information to disk for the specified driver

### Declaration

```
ibool NAPI GA_saveModeProfile(
    GA_devCtx *dc,
    GA_modeProfile *profile)
```

### Prototype In

snap/graphics.h

### Parameters

<i>dc</i>	Device context to save mode profile information for
<i>profile</i>	Mode profile information block to save

### Return Value

True on success, false on error (see *GA\_status* for error info)

### Description

This function saves the mode profile information to disk. The mode profile is the list of modes that the driver will report to user applications. You can modify this list to customise the set of display modes that the driver supports.

---

**Note:** *This function does not update the mode profile in the driver itself, but rather you should call *SetModeProfile* to update the internal mode profile first before saving the options to disk.*

---

### See Also

*GetModeProfile*, *SetModeProfile*, *GA\_saveOptions*, *GA\_saveCRTCTimings*, *GA\_loadModeProfile*

## GA\_saveMonitorInfo

Save the monitor profile to disk for the specified driver

### Declaration

```
ibool NAPI GA_saveMonitorInfo(  
    GA_devCtx *dc,  
    GA_monitor *monitor)
```

### Prototype In

snap/graphics.h

### Parameters

<i>dc</i>	Device context to save options information for
<i>monitor</i>	Monitor profile to save to disk

### Return Value

True on success, false on error (see *GA\_status* for error info)

### Description

This function saves the monitor profile information to disk.

---

**Note:** *This function does not update the driver monitor stored in the driver itself, but rather you should call SetMonitorInfo to update the internal options first before saving the options to disk.*

---

### See Also

GetMonitorInfo, SetMonitorInfo



## GA\_saveOptions

Save the options to disk for the specified driver

### Declaration

```
ibool NAPI GA_saveOptions(
    GA_devCtx *dc,
    GA_options *options)
```

### Prototype In

snap/graphics.h

### Parameters

<i>dc</i>	Device context to save options information for
<i>options</i>	Driver options information block to save

### Return Value

True on success, false on error (see *GA\_status* for error info)

### Description

This function saves the driver options information to disk.

---

**Note:** *This function does not update the driver options in the driver itself, but rather you should call *SetOptions* to update the internal options first before saving the options to disk.*

---

### See Also

GetOptions, SetOptions, *GA\_saveModeProfile*, *GA\_saveCRTCTimings*

## GA\_sectRect

Compute the intersection between two rectangles.

### Declaration

```
ibool GA_sectRect(  
    GA_rect r1,  
    GA_rect r2,  
    GA_rect *d)
```

### Prototype In

snap/graphics.h

### Parameters

<i>r1</i>	First rectangle to intersect
<i>r2</i>	Second rectangle to intersect
<i>d</i>	Place to store the resulting intersection

### Return Value

True if the rectangles intersect, false if not.

### Description

Computes the intersection of two rectangles, and returns the result in a third. If the rectangles actually intersect (the intersection is not an empty rectangle) then this function returns true, otherwise it will return false.

### See Also

*GA\_sectRectFast*, *GA\_sectRectCoord*, *GA\_unionRect*

## GA\_sectRectCoord

Compute the intersection between two rectangles.

### Declaration

```
ibool GA_sectRectCoord(
    int left1,
    int top1,
    int right1,
    int bottom1,
    int left2,
    int top2,
    int right2,
    int bottom2,
    GA_rect *d)
```

### Prototype In

snap/graphics.h

### Parameters

<i>left1</i>	Left coordinate of first rectangle to intersect
<i>top1</i>	Top coordinate of first rectangle to intersect
<i>right1</i>	Right coordinate of first rectangle to intersect
<i>bottom1</i>	Bottom coordinate of first rectangle to intersect
<i>left2</i>	Left coordinate of second rectangle to intersect
<i>top2</i>	Top coordinate of second rectangle to intersect
<i>right2</i>	Right coordinate of second rectangle to intersect
<i>bottom2</i>	Bottom coordinate of second rectangle to intersect
<i>d</i>	Place to store the resulting intersection

### Return Value

True if the rectangles intersect, false if not.

### Description

Computes the intersection of two rectangles, and returns the result in a third. If the rectangles actually intersect (the intersection is not an empty rectangle) then this function returns true, otherwise it will return false.

### See Also

*GA\_sectRectFastCoord*, *GA\_sectRect*, *GA\_unionRect*

## GA\_sectRectFast

Compute the intersection between two rectangles.

### Declaration

```
void GA_sectRectFast(  
    GA_rect r1,  
    GA_rect r2,  
    GA_rect *d)
```

### Prototype In

snap/graphics.h

### Parameters

<i>r1</i>	First rectangle to intersect
<i>r2</i>	Second rectangle to intersect
<i>d</i>	Place to store the resulting intersection

### Description

This is the same as *GA\_sectRect* but it is implemented as a macro and does not test the rectangle for intersection.

### See Also

*GA\_sectRect*, *GA\_sectRectCoord*, *GA\_unionRect*

## GA\_sectRectFastCoord

Compute the intersection between two rectangles.

### Declaration

```
ibool GA_sectRectFastCoord(
    int left1,
    int top1,
    int right1,
    int bottom1,
    int left2,
    int top2,
    int right2,
    int bottom2,
    GA_rect *d)
```

### Prototype In

snap/graphics.h

### Parameters

<i>left1</i>	Left coordinate of first rectangle to intersect
<i>top1</i>	Top coordinate of first rectangle to intersect
<i>right1</i>	Right coordinate of first rectangle to intersect
<i>bottom1</i>	Bottom coordinate of first rectangle to intersect
<i>left2</i>	Left coordinate of second rectangle to intersect
<i>top2</i>	Top coordinate of second rectangle to intersect
<i>right2</i>	Right coordinate of second rectangle to intersect
<i>bottom2</i>	Bottom coordinate of second rectangle to intersect
<i>d</i>	Place to store the resulting intersection

### Return Value

True if the rectangles intersect, false if not.

### Description

Computes the intersection of two rectangles, and returns the result in a third. If the rectangles actually intersect (the intersection is not an empty rectangle) then this function returns true, otherwise it will return false.

### See Also

*GA\_sectRectFastCoord*, *GA\_sectRect*, *GA\_unionRect*

## GA\_setActiveDevice

Sets the active display device for multiple display devices

### Declaration

```
ibool NAPI GA_setActiveDevice(
    N_int32 deviceIndex)
```

### Prototype In

snap/graphics.h

### Parameters

*deviceIndex*                      Index of device to make active

### Return Value

True on success, false on error.

### Description

This function allows the application to switch between the primary and secondary display controllers, making one of the controllers the active controller. When a controller is the active controller it will function as though it is the only display controller in the system. You can also select a 'mixed' mode of operation by passing in the `DEVICE_MIXED_MODED` parameter, which fully enables the primary display controller but only enables the secondary display controllers memory mappings. If mixed mode is selected, it is the responsibility of the *calling* application to ensure that the secondary controller's VGA memory spaces are disabled *before* setting this mode to avoid conflicts between the controllers.

The device numbering starts at 0 for the primary display controller, and increments by one for each supports display controller. Ie: the second controller is device 1 while the third controller is device 2 etc. There is no limit of the number of devices supported, other than the number of PCI/ AGP slots available in a particular machine.

### See Also

*GA\_loadDriver*

## GA\_setCRTCTimings

Set the CRTC timings for a specific display mode and refresh rate

### Declaration

```
ibool NAPI GA_setCRTCTimings(
    GA_devCtx *dc,
    GA_modeInfo *modeInfo,
    GA_CRTCInfo *crtc,
    ibool makeDefault)
```

### Prototype In

snap/graphics.h

### Parameters

<i>dc</i>	Device context to save CRTC timings information for
<i>modeInfo</i>	Display mode information block
<i>crtc</i>	CRTC timings to update for the mode
<i>makeDefault</i>	True to make the timings the default for the mode

### Return Value

True on success, false on error.

### Description

This function updates the CRTC timings for a particular display mode and refresh rate (refresh rate stored in the *crtc* parameter) in the 7internal CRTC timings database. If the resolution and/or refresh rate is not supported by the display mode, this function returns false.

The *makeDefault* parameter is used to determine whether the CRTC timings should be made the default timings for the display mode. If this parameter is true, the passed in timings will become the default timings for that display mode. If not, the timings will be updated but the default refresh rate will not be changed for the display mode.

### See Also

*GA\_saveCRTCTimings*, *GA\_restoreCRTCTimings*, *SetCRTCTimings*

## GA\_setDefaultRefresh

Set the default refresh rate for a specific resolution and color depth

### Declaration

```
ibool NAPI GA_setDefaultRefresh(
    GA_devCtx *dc,
    N_int32 xRes,
    N_int32 yRes,
    N_int32 bits,
    N_int32 refreshRate,
    ibool saveToDisk)
```

### Prototype In

snap/graphics.h

### Parameters

<i>dc</i>	Device context to save CRTIC timings information for
<i>xRes</i>	X resolution for the mode to change
<i>yRes</i>	Y resolution for the mode to change
<i>bits</i>	Color depth of the mode to change (0 for text modes)
<i>refreshRate</i>	New refresh rate to make the default
<i>saveToDisk</i>	True to save the changes permanently to disk

### Return Value

True on success, false on error.

### Description

This function sets the default refresh rate for a specific resolution and color depth to the passed in refresh rate for the mode.

---

**Note:** *Internally the CRTIC timings are maintained independantly of specific colors depths, so changing the default refresh rate for 640x480x8 also changes it for all other 640x480 modes. The main reason we have the 'bits' parameter is so we can tell when we are dealing with text modes which are handled slightly differently.*

---

### See Also

GA\_saveCRTCTimings, GA\_restoreCRTCTimings



## GA\_setGlobalOptions

Sets the current global options for the graphics device driver.

### Declaration

```
void NAPI GA_setGlobalOptions(  
    GA_globalOptions *options)
```

### Prototype In

snap/graphics.h

### Parameters

*options*                      Global options to make active.

### Description

This function installs a new set of global device options that control the operation of all the devices at runtime. A default set of options is always built into the device driver, but the options can be changed at any time. The options may be made permanent with a call to the *GA\_saveGlobalOptions* function.

---

**Note:** *The dwSize member of the profile structure is intended for future compatibility, and must be set to the size of the structure before calling this function. Only the number of bytes set in the dwSize member will be copied from the callers structure.*

---

### See Also

*GA\_getGlobalOptions, GA\_saveGlobalOptions*

## GA\_setMinimumDriverVersion

Sets the minimum driver version supported

### Declaration

```
void NAPI GA_setMinimumDriverVersion(
    N_uint32 version,
    N_int32 allowFallback,
    N_int32 shared)
```

### Prototype In

snap/graphics.h

### Parameters

<i>version</i>	Minimum driver version to accept
<i>allowFallback</i>	Allow the fallback drivers to load
<i>shared</i>	True if the device driver should be loaded into shared memory

### Description

This function can be used to exclude early device driver versions if they are known not to have the necessary feature support from being loaded. This function will stop the drivers from loading early in the driver load process, so as to avoid any problems that might crop up during loading and initialisation.

The allowFallback parameter is used to allow the loading of the VBE/Core and VGA fallback drivers. If this parameter is false, then the VBE/Core and VGA drivers will fail to load. If this parameter is not false then they will load as normal.

---

**Note:** *The default minimum version is 0 and the default value for allowFallback is true.*

---

### See Also

*GA\_loadDriver*

## GA\_softStereoExit

Cleans up and exits the software stereo module.

### Declaration

```
void NAPI GA_softStereoExit(void);  
STEREO_ENTRY void NAPI GA_softStereoExit(void)
```

### Prototype In

snap/graphics.h

### Description

This function cleans up and exits the software stereo page flipping module, and restores all interrupt handlers hooked by the driver.

### See Also

*GA\_softStereoInit*

## GA\_softStereoGetFlipStatus

Returns the status of the last scheduled stereo display start change.

### Declaration

```
N_int32          NAPI GA_softStereoGetFlipStatus(void);  
STEREO_ENTRY N_int32 NAPI GA_softStereoGetFlipStatus(void)
```

### Prototype In

snap/graphics.h

### Return Value

-1 if flipper is not running at all; 0 if flip has not occurred; 1 if flip has occurred and currently on left eye; 2 if flip has occurred and currently on right eye.

### Description

This function returns the status of the last scheduled software stereo display start change. This bit is sticky and is set to 0 when the *GA\_softStereoScheduleFlip* function is called, and reset to 1 when the flip actually occurs.

This function also serves to inform the calling application whether the flipper is already running or not. This will help insure that the flipper will only be used one instance at a time.

### See Also

*GA\_softStereoScheduleFlip*, *GA\_softStereoWaitTillFlipped*

## GA\_softStereoInit

Initialises the software stereo page flipping module.

### Declaration

```
ibool          NAPI GA_softStereoInit(GA_devCtx *dc);
STEREO_ENTRY ibool NAPI GA_softStereoInit(
    GA_devCtx *dc)
```

### Prototype In

snap/graphics.h

### Parameters

*dc*                SNAP device context to use to access the hardware

### Description

This function initialises the software stereo page flip handler, and does one time initialisation and calibration. The calibration will take about 1 second for most resolutions and refresh rates.

---

**Note:** This function ***must*** be called after the system has already been put into graphics mode, since it does calibration based on the currently active display mode and refresh rate.

**Note:** Software stereo page flipping is off by default when this function is called, and you must call `GA_softStereoOn` to activate it.

---

### See Also

`GA_softStereoExit`, `GA_softStereoOn`

## GA\_softStereoOff

Turns off the automatic software stereo page flipping.

### Declaration

```
void NAPI GA_softStereoOff(void);  
STEREO_ENTRY void NAPI GA_softStereoOff(void)
```

### Prototype In

snap/graphics.h

### Description

This function turns off the software stereo page flipping, putting the system back into 2D mode and turning off the LC shutter glasses.

### See Also

*GA\_softStereoExit, GA\_softStereoOn*

## GA\_softStereoOn

Enables software stereo page flipping.

### Declaration

```
void NAPI GA_softStereoOn(void);  
STEREO_ENTRY void NAPI GA_softStereoOn(void)
```

### Prototype In

snap/graphics.h

### Description

This function enables the software stereo page flipping, and starts the real time clock counter running in the background.

### See Also

*GA\_softStereoScheduleFlip, GA\_softStereoGetFlipStatus, GA\_softStereoOff*

## GA\_softStereoScheduleFlip

Schedule a new stereo display start address change.

### Declaration

```
void NAPI GA_softStereoScheduleFlip(N_uint32 leftAddr, N_uint32
rightAddr);
STEREO_ENTRY void NAPI GA_softStereoScheduleFlip(
    N_uint32 leftAddr,
    N_uint32 rightAddr)
```

### Prototype In

snap/graphics.h

### Parameters

<i>leftAddr</i>	Left display start address to make active
<i>rightAddr</i>	Right display start address to make active

### Description

This function schedules a new left and right start address to take hold on the next vertical retrace for the LC shutter glasses. This is used for double and triple buffering when stereo mode is active.

### See Also

*GA\_softStereoGetFlipStatus, GA\_softStereoWaitTillFlipped*



## GA\_softStereoWaitTillFlipped

Waits until the last scheduled flip operation has occurred.

### Declaration

```
void NAPI GA_softStereoWaitTillFlipped(void);  
STEREO_ENTRY void NAPI GA_softStereoWaitTillFlipped(void)
```

### Prototype In

snap/graphics.h

### Description

This function polls the status of the last scheduled flip operation, and waits until it has completed. However this function also does the important job of checking the RTC clock to ensure that it continues to run, and will re-start it if it has stopped for some reason. Hence you should call this function when you need to spin until the flip has occurred to catch problems when the RTC clock stops running.

### See Also

*GA\_softStereoScheduleFlip, GA\_softStereoGetFlipStatus*

## GA\_status

Returns the error code for *GA\_loadDriver* if the function failed.

### Declaration

```
int NAPI GA_status(void)
```

### Prototype In

snap/graphics.h

### Return Value

Status code for the *GA\_loadDriver* function.

### Description

Returns the error code for *GA\_loadDriver* if the function failed.

### See Also

*GA\_errorMsg*

## GA\_unionRect

Computes the union of two rectangles.

### Declaration

```
void GA_unionRect(  
    GA_rect r1,  
    GA_rect r2,  
    GA_rect *d)
```

### Prototype In

snap/graphics.h

### Parameters

<i>r1</i>	First rectangle to compute union of
<i>r2</i>	Second rectangle to compute union of
<i>d</i>	Place to store resulting union rectangle

### Description

This function computes the union of two rectangles, and stores the result in a third rectangle.

### See Also

*GA\_unionRectCoord*, *GA\_sectRect*

## GA\_unionRectCoord

Computes the union of two rectangles.

### Declaration

```
void GA_unionRectCoord(
    int left1,
    int top1,
    int right1,
    int bottom1,
    int left2,
    int top2,
    int right2,
    int bottom2,
    GA_rect *d)
```

### Prototype In

snap/graphics.h

### Parameters

<i>left1</i>	Left coordinate of first rectangle to compute union of
<i>top1</i>	Top coordinate of first rectangle to compute union of
<i>right1</i>	Right coordinate of first rectangle to compute union of
<i>bottom1</i>	Bottom coordinate of first rectangle to compute union of
<i>left2</i>	Left coordinate of second rectangle to compute union of
<i>top2</i>	Top coordinate of second rectangle to compute union of
<i>right2</i>	Right coordinate of second rectangle to compute union of
<i>bottom2</i>	Bottom coordinate of second rectangle to compute union of
<i>d</i>	Place to store resulting union rectangle

### Description

This function computes the union of two rectangles, and stores the result in a third rectangle.

### See Also

*GA\_sectRect*

## GA\_unloadDriver

Unloads the loaded graphics device driver.

### Declaration

```
void NAPI GA_unloadDriver(  
    GA_devCtx *dc)
```

### Prototype In

snap/graphics.h

### Parameters

*dc*                      Pointer to device context to unload

### Description

This function simply unloads the graphics driver from memory, and frees any of the internal resources that have been allocated for it. Specifically it will free any locked memory and any interrupt handlers hooked for the driver.

### See Also

*GA\_loadDriver*

## GA\_unloadRef2d

Unloads the 2D reference rasteriser device driver chain

### Declaration

```
void NAPI GA_unloadRef2d(  
    GA_devCtx _FAR_ *dc)
```

### Prototype In

snap/graphics.h

### Parameters

*dc*                      Pointer to SNAP device context to unload driver chain for

### Description

This function unloads the SNAP 2D Reference Rasteriser from memory, without unloading the actual hardware device driver. This can be used to unload the reference rasteriser and then reload a new copy such as when changing color depths dynamically within a display driver.

### See Also

*GA\_unloadDriver*

## GA\_unloadRegionMgr

Unloads the 2D region manager driver.

### Declaration

```
void NAPI GA_unloadRegionMgr(  
    PE_MODULE *hModule)
```

### Prototype In

snap/graphics.h

### Parameters

*hModule*                      Pointer to the module handle for the driver to unload

### Description

This function unloads the SNAP 2D region manager from memory. This should only be called if the application has loaded a separate set of SNAP region manager functions with *GA\_loadRegionMgr*.

### See Also

*GA\_loadRegionMgr*

## GA\_useDoubleScan

Determine if a mode should use double scanning or not.

### Declaration

```
ibool NAPI GA_useDoubleScan(  
    GA_modeInfo *modeInfo)
```

### Prototype In

snap/graphics.h

### Parameters

*modeInfo*                      Mode information for mode to check

### Return Value

True if double scan should be used, false if not.

### Description

Determines if we should use a double scan mode for the graphics mode. This is true if we have a low resolution mode and the mode supports double scanning.



## MCS\_begin

Checks to see if DDC/CI communication is available and opens it

### Declaration

```
int NAPI MCS_begin(  
    GA_devCtx *dc)
```

### Prototype In

snap/ddc.h

### Parameters

*dc*            SNAP device driver to use for communications

### Return Value

0 if DDC available, 1 if not available, -1 if unable to communicate via DDC

### Description

Obsolete. Use *MCS\_beginExt* instead.

### See Also

*MCS\_beginExt*

## MCS\_beginExt

Checks to see if DDC/CI communication is available and opens it

### Declaration

```
int NAPI MCS_beginExt(
    GA_devCtx *dc,
    N_int32 channel)
```

### Prototype In

snap/ddc.h

### Parameters

<i>dc</i>	SNAP device driver to use for communications
<i>channel</i>	I2C channel to use to control the monitor (0 for primary monitor)

### Return Value

One of the *DDC\_errCode* error return values.

### Description

This function initializes the DDC communications module. After checking that the I2C interface is working, this function will attempt to initialize the DDC communication channel and verify that DDC2B communication is possible. If DDC2B is not available, this function returns a value of *ddcNoCommunication* (ie: cannot communicate with slave). The most likely cause of this failure condition is that there is no DDC capable monitor attached to the graphics device.

Once DDC2B functionality has been detected, this function attempts to read the capabilities string from the monitor using the DDC/CI protocol. If a DDC/CI compatible monitor is successfully found, this function returns *ddcOk*. If the capabilities string could not be read, this function returns *ddcNotAvailable*, which generally indicates that the attached monitor is not DDC/CI capable. This function also parses the monitor capabilities string to determine what features the monitor supports and initializes the device driver accordingly.

---

**Note:** *After you have finished with communications over the DDC/CI interface, you must call MCS\_end to close the communications channel.*

---

### See Also

*MCS\_end*

## MCS\_enableControl

Enables or disables a specific MCCA control

### Declaration

```
ibool NAPI MCS_enableControl(
    uchar controlCode,
    N_int32 enable)
```

### Prototype In

snap/ddc.h

### Parameters

<i>controlCode</i>	MCCA control code ( <i>MCS_controlsType</i> )
<i>enable</i>	1 to enable, 0 to disable the control

### Return Value

True if the function succeeded, false if it failed.

### Description

This function enables or disables a specified MCCA control for the attached monitor. This intention of this function is for controls that have an enabled or disabled state, such as a microphone input, S-Video camera input or other such controls. Disabling the control effectively turns off the input, and enabling it turns it on again. Note that most controls like audio volume controls can be fully controlled via the regular *MCS\_setControlValue* function.

### See Also

*MCS\_isControlSupported*, *MCS\_getControlValue*, *MCS\_setControlValue*, *MCS\_resetControl*, *MCS\_saveCurrentSettings*

## MCS\_end

Closes the DDC/CI communications channel

### Declaration

```
void NAPI MCS_end(void)
```

### Prototype In

snap/ddc.h

### Description

This function closes the DDC/CI communications channel and must be called after you have finished DDC/CI communications.

### See Also

*MCS\_begin*

## MCS\_getCapabilitiesString

Reads the DDC/CI capabilities string from the attached monitor.

### Declaration

```
int NAPI MCS_getCapabilitiesString(
    char *data,
    N_int32 maxlen)
```

### Prototype In

snap/ddc.h

### Parameters

<i>data</i>	Buffer to read capabilities string into
<i>maxlen</i>	Maximum length of the buffer to receive data

### Return Value

Length of the capabilities string read from monitor, or -1 on error.

### Description

This function reads the DDC/CI capabilities string from the attached monitor and returns it in the buffer passed in the data parameter. This function will only read up to maxlen characters from the monitor, and it will return the actual number of characters read from the monitor. For a description of what the capabilities string contains, please refer to the VESA DDC/CI specification or the ACCESS.bus specification.

---

**Note:** *This function removes the binary EDID data that may be present in the capabilities string before returning it, so that the string may be viewed entirely as an ASCII string. The EDID data can be easily read with the DDC\_readEDID function so it is superfluous to include the EDID in the capabilities string.*

---

## MCS\_getControlMax

Returns the maximum value for an MCCS control

### Declaration

```
ibool NAPI MCS_getControlMax(
    uchar controlCode,
    ushort *max)
```

### Prototype In

snap/ddc.h

### Parameters

<i>controlCode</i>	MCCS control code ( <i>MCS_controlsType</i> )
<i>max</i>	Place to store the maximum value for the control

### Return Value

True if the function succeeded, false if it failed.

### Description

This function obtains the maximum value for a specified MCCS control for the attached monitor. Continuous controls may accept any value between zero and the maximum value, specific to each control. Non-continuous controls can only accept specific values. The maximum value returned for non-continuous controls represents the maximum index for the acceptable values for the control. A value of 0 for non-continuous controls represents that no value has been selected.

---

**Note:** *This function internally reads the maximum value from the attached monitor only once for each control between an MCS\_begin and MCS\_end block. If you call it again after the maximum value for a control has been read, it simply returns an internal cached value for maximum performance.*

---

### See Also

*MCS\_isControlSupported, MCS\_enableControl, MCS\_getControlValue, MCS\_setControlValue, MCS\_resetControl, MCS\_saveCurrentSettings*

## MCS\_getControlValue

Returns the current value for an MCCS control

### Declaration

```
ibool NAPI MCS_getControlValue(
    uchar controlCode,
    ushort *value)
```

### Prototype In

snap/ddc.h

### Parameters

<i>controlCode</i>	MCCS control code ( <i>MCS_controlsType</i> )
<i>value</i>	Place to store the current value for the control

### Return Value

True if the function succeeded, false if it failed.

### Description

This function obtains the current value for a specified MCCS control for the attached monitor. Continuous controls may accept any value between zero and the maximum value, specific to each control. Non-continuous controls can only accept specific values. A value of 0 for non-continuous controls represents that no value has been selected.

---

**Note:** *If you need to retrieve the values of multiple controls at a time, you should use the more efficient MCS\_getControlValues function which can read multiple control values directly from the monitor in a single packet.*

---

### See Also

*MCS\_isControlSupported, MCS\_enableControl, MCS\_getControlValues, MCS\_setControlValue, MCS\_setControlValues, MCS\_resetControl, MCS\_saveCurrentSettings*

## MCS\_getControlValues

Returns the current value for a list of MCCC controls

### Declaration

```
ibool NAPI MCS_getControlValues(
    N_int32 numControls,
    uchar *controlCodes,
    ushort *values)
```

### Prototype In

snap/ddc.h

### Parameters

<i>numControls</i>	Number of control values to read (40 max)
<i>controlCodes</i>	Array of MCCC control codes ( <i>MCS_controlsType</i> )
<i>values</i>	Array to store the current value for each control

### Return Value

True if the function succeeded, false if it failed.

### Description

This function gets the current value for multiple MCCC controls at a time. This function is similar to *MCS\_getControlValue*, but is much faster for reading the value of multiple controls due to the reduced traffic over the I2C bus.

---

**Note:** *This function can only read up to 40 controls at a time. If you need to read more than 40 controls, you will need to call this function mutiple times.*

---

### See Also

*MCS\_isControlSupported, MCS\_enableControl, MCS\_getControlValue, MCS\_setControlValue, MCS\_setControlValues, MCS\_resetControl, MCS\_saveCurrentSettings*



## MCS\_getSelfTestReport

Reads the self test report from the attached monitor

### Declaration

```
ibool NAPI MCS_getSelfTestReport (  
    uchar *flags,  
    uchar *data,  
    uchar *length)
```

### Prototype In

snap/ddc.h

### Parameters

<i>flags</i>	Place to store the mode flags
<i>data</i>	Place to store the data read from the display
<i>length</i>	Place to store the length of the data read

### Return Value

True if the function succeeded, false if it failed.

### Description

This function gets the monitor to perform a self test operation and report the results back to the host. The information reported back is monitor vendor specific, so this function is generally used by monitor vendors to read self test information from monitors in the field for diagnostics and reporting.

## MCS\_getTimingReport

Reads the current horizontal and vertical frequency from the monitor

### Declaration

```
ibool NAPI MCS_getTimingReport(
    uchar *flags,
    ushort *hFreq,
    ushort *vFreq)
```

### Prototype In

snap/ddc.h

### Parameters

<i>flags</i>	Place to store the mode flags
<i>hFreq</i>	Place to store the horizontal frequency (kHz * 100)
<i>vFreq</i>	Place to store the vertical frequency (Hz * 100)

### Return Value

True if the function succeeded, false if it failed.

### Description

This function reads the horizontal and vertical frequencies for the current display mode from the monitor. The horizontal frequency is reported in units of kHz \* 100 (ie: a value of 3150 is 31.5 KHz). The vertical frequency is reported in units of Hz \* 100 (ie: a value of 7500 is 75 Hz).

This function also reads the sync polarities and returns them in the flags parameter. The values returned in the flags parameter are defined by the *MCS\_polarityFlagsType* enumeration.

---

**Note:** *It is highly recommended that all monitor vendors implement this DDC/CI function, since this function contains useful feedback information to the user and graphics device drivers.*

---

## MCS\_isControlSupported

Checks to see if an MCCS control is supported

### Declaration

```
ibool NAPI MCS_isControlSupported(  
    uchar controlCode)
```

### Prototype In

snap/ddc.h

### Parameters

*controlCode*                      MCCS control code to check (*MCS\_controlsType*)

### Return Value

True if the control is supported, false if not.

### Description

This function verifies that the specified MCCS control is supported by the attached monitor. If the control is supported this function returns true, otherwise it returns false.

### See Also

*MCS\_enableControl*, *MCS\_getControlValue*, *MCS\_setControlValue*, *MCS\_resetControl*,  
*MCS\_saveCurrentSettings*

## MCS\_resetControl

Resets the value of an MCCS control to the factory default setting

### Declaration

```
ibool NAPI MCS_resetControl(  
    uchar controlCode)
```

### Prototype In

snap/ddc.h

### Parameters

*controlCode*                      MCCS control code (*MCS\_controlsType*)

### Return Value

True if the function succeeded, false if it failed.

### Description

This function resets the value for a specified MCCS control to the factory default setting. This can be used to reset values to reasonable settings if the user has completely tweaked them out of control.

### See Also

*MCS\_isControlSupported*, *MCS\_enableControl*, *MCS\_getControlValue*,  
*MCS\_setControlValue*, *MCS\_saveCurrentSettings*

## MCS\_saveCurrentSettings

Saves the current settings in the monitor NVRAM

### Declaration

```
ibool NAPI MCS_saveCurrentSettings(void)
```

### Prototype In

snap/ddc.h

### Return Value

True if the function succeeded, false if it failed.

### Description

This function saves the current settings for all controls in the non-volatile RAM (NVRAM) of the attached monitor. This essentially tells the monitor to remember all the current settings for the current display mode in the NVRAM in the monitor.

### See Also

*MCS\_isControlSupported, MCS\_enableControl, MCS\_getControlValue, MCS\_setControlValue, MCS\_resetControl*

## MCS\_setControlValue

Sets the value of an MCCS control to the specified value

### Declaration

```
ibool NAPI MCS_setControlValue(
    uchar controlCode,
    ushort value)
```

### Prototype In

snap/ddc.h

### Parameters

<i>controlCode</i>	MCCS control code ( <i>MCS_controlsType</i> )
<i>value</i>	New value for the control

### Return Value

True if the function succeeded, false if it failed.

### Description

This function sets the current value for a specified MCCS control to the specified value. Continuous controls may accept any value between zero and the maximum value, specific to each control. Non-continuous controls can only accept specific values. The maximum value returned for non-continuous controls represents the maximum index for the acceptable values for the control. A value of 0 for non-continuous controls represents that no value has been selected.

---

**Note:** *If you need to set the values of multiple controls at a time, you should use the more efficient MCS\_setControlValues function which can set up to 40 control values at the same time.*

---

### See Also

*MCS\_isControlSupported, MCS\_enableControl, MCS\_getControlValue, MCS\_resetControl, MCS\_saveCurrentSettings*

## MCS\_setControlValues

Sets the value of multiple MCCS controls at a time

### Declaration

```
ibool NAPI MCS_setControlValues(
    N_int32 numControls,
    uchar *controlCodes,
    ushort *values)
```

### Prototype In

snap/ddc.h

### Parameters

<i>numControls</i>	Number of control values to program (40 max)
<i>controlCodes</i>	Array of MCCS control codes ( <i>MCS_controlsType</i> )
<i>values</i>	Array of new values for the control

### Return Value

True if the function succeeded, false if it failed.

### Description

This function sets the current value for multiple MCCS controls at a time. This function is similar to *MCS\_setControlValue*, but is much faster for setting the value of multiple controls due to reduced traffic over the I2C bus.

---

**Note:** *This function can only program up to 40 controls at a time. If you need to program more than 40 controls, you will need to call this function multiple times.*

---

### See Also

*MCS\_isControlSupported, MCS\_enableControl, MCS\_getControlValue, MCS\_getControlValues, MCS\_setControlValue, MCS\_resetControl, MCS\_saveCurrentSettings*

## MDBX\_close

Close the monitor database.

### Declaration

```
void NAPI MDBX_close(void)
```

### Prototype In

snap/monitor.h

### Description

Closes the monitor database when done.

### See Also

*MDBX\_open*



## **MDBX\_first**

Seeks to the first record in the monitor database

### **Declaration**

```
int NAPI MDBX_first(  
    GA_monitor *rec)
```

### **Prototype In**

snap/monitor.h

### **Parameters**

*rec*                      Monitor record to store entry in

### **Return Value**

Error code for operation.

### **Description**

Attempts to seek to the first entry in the monitor database. If the record exists, the parameters *rec* is filled in with the monitor information.

### **See Also**

*MDBX\_last, MDBX\_next, MDBX\_prev*

## MDBX\_flush

Flushes the monitor database to disk.

### Declaration

```
int NAPI MDBX_flush(void)
```

### Prototype In

snap/monitor.h

### Return Value

Error code for operation.

### Description

Flushes the database records to disk after they have been updated.

### See Also

*MDBX\_first, MDBX\_insert, MDBX\_update*

## **MDBX\_getErrCode**

Return the current monitor database error code.

### **Declaration**

```
int NAPI MDBX_getErrCode(void)
```

### **Prototype In**

snap/monitor.h

### **Return Value**

Monitor database error code.

### **Description**

This function return the current monitor database error code for the previous operation.

### **See Also**

*MDBX\_getErrorMsg*

## MDBX\_getErrorMsg

Return the current monitor database error code as a string.

### Declaration

```
const char * NAPI MDBX_getErrorMsg(void)
```

### Prototype In

snap/monitor.h

### Return Value

Monitor database error code as a string.

### Description

This function return the current monitor database error code for the previous operation as a string.

---

**Note:** *The string is returned in a temporary buffer that is reused. Hence the calling application should copy the returned string to a different buffer immediately after calling this function and before calling any other SNAP functions.*

---

### See Also

*MDBX\_getErrCode*

## MDBX\_importINF

Import a Windows INF file into the monitor database

### Declaration

```
int NAPI MDBX_importINF(  
    const char *filename,  
    char *mfr)
```

### Prototype In

snap/monitor.h

### Parameters

<i>filename</i>	Path to the Windows INF file to parse and import
<i>mfr</i>	Place to store first manufacturer string read (NULL to ignore)

### Return Value

Error code for operation.

### Description

This function is used to parse a Windows INF file from disk and insert all monitor records into the monitor database. This can be used to implement a monitor 'Have Disk' option for any supported Operating System.

---

**Note:** *The monitor database is expected to be opened before this function is called, and should be flushed and closed when done. Hence this function can also import multiple INF files at a time into the database.*

---

## MDBX\_insert

Inserts a new record into the monitor database

### Declaration

```
int NAPI MDBX_insert(  
    GA_monitor *rec)
```

### Prototype In

snap/monitor.h

### Parameters

*rec*                      Monitor record to store entry in

### Return Value

Error code for operation.

### Description

Inserts the specified record into the database. Because the database's are always very small, we simply do a complete copy of the record array, inserting the new record along the way. If the record already exists, we update the existing record with the new data.

### See Also

*MDBX\_first, MDBX\_update, MDBX\_flush*

## MDBX\_last

Seeks to the last record in the monitor database

### Declaration

```
int NAPI MDBX_last(  
    GA_monitor *rec)
```

### Prototype In

snap/monitor.h

### Parameters

*rec*                      Monitor record to store entry in

### Return Value

Error code for operation.

### Description

Attempts to seek to the last entry in the monitor database. If the record exists, the parameters *rec* is filled in with the monitor information.

### See Also

*MDBX\_first*, *MDBX\_next*, *MDBX\_prev*

## MDBX\_next

Seeks to the next record in the monitor database

### Declaration

```
int NAPI MDBX_next(  
    GA_monitor *rec)
```

### Prototype In

snap/monitor.h

### Parameters

*rec*                      Monitor record to store entry in

### Return Value

Error code for operation.

### Description

Attempts to seek to the next entry in the monitor database after the passed in record. If the record exists, the parameters *rec* is filled in with the monitor information.

### See Also

*MDBX\_first*, *MDBX\_last*, *MDBX\_prev*



## **MDBX\_open**

Open the monitor database.

### **Declaration**

```
ibool NAPI MDBX_open(  
    const char *filename)
```

### **Prototype In**

snap/monitor.h

### **Parameters**

*filename*                      Path to monitor database to open

### **Return Value**

Error code for operation.

### **Description**

Attempts to open the monitor database. This may fail if the file was not found or the file was not a valid database file.

### **See Also**

*MDBX\_close*

## MDBX\_prev

Seeks to the previous record in the monitor database

### Declaration

```
int NAPI MDBX_prev(  
    GA_monitor *rec)
```

### Prototype In

snap/monitor.h

### Parameters

*rec*                      Monitor record to store entry in

### Return Value

Error code for operation.

### Description

Attempts to seek to the previous entry in the monitor database after the passed in record. If the record exists, the parameters *rec* is filled in with the monitor information.

### See Also

*MDBX\_first*, *MDBX\_last*, *MDBX\_next*

## MDBX\_update

Updates the current record into the monitor database

### Declaration

```
int NAPI MDBX_update(  
    GA_monitor *rec)
```

### Prototype In

snap/monitor.h

### Parameters

*rec*                      Monitor record to store entry in

### Return Value

Error code for operation.

### Description

Updates the current record with the new values in the passed in record.

### See Also

*MDBX\_first, MDBX\_insert, MDBX\_flush*

## PE\_freeLibrary

Frees a loaded Portable Binary DLL

### Declaration

```
void WINAPI PE_freeLibrary(  
    PE_MODULE *hModule)
```

### Prototype In

drvlib/peloder.h

### Parameters

*hModule*                      Handle to a loaded PE DLL library to free

### Description

This function frees a loaded PE DLL library from memory.

### See Also

*PE\_getProcAddress*, *PE\_loadLibrary*

## PE\_getError

Returns the error code for the last operation

### Declaration

```
int WINAPI PE_getError(void)
```

### Prototype In

drvlib/peloder.h

### Return Value

Error code for the last operation.

### See Also

*PE\_getProcAddress, PE\_loadLibrary*

## PE\_getFileSize

Find the actual size of a PE file image

### Declaration

```
ulong WINAPI PE_getFileSize(  
    FILE *f,  
    ulong startOffset)
```

### Prototype In

drvlib/peloder.h

### Parameters

<i>f</i>	Handle to open file to read driver from
<i>startOffset</i>	Offset to the start of the driver within the file

### Return Value

Size of the DLL file on disk, or -1 on error

### Description

This function scans the headers for a Portable Binary DLL to determine the length of the DLL file on disk.

## PE\_getProcAddress

Gets a function address from a Portable Binary DLL

### Declaration

```
void * WINAPI PE_getProcAddress(  
    PE_MODULE *hModule,  
    const char *szProcName)
```

### Prototype In

drvlib/peloder.h

### Parameters

<i>hModule</i>	Handle to a loaded PE DLL library
<i>szProcName</i>	Name of the function to get the address of

### Return Value

Pointer to the function, or NULL on failure.

### Description

This function searches for the named, exported function in a loaded PE DLL library, and returns the address of the function. If the function is not found in the library, this function return NULL.

### See Also

*PE\_loadLibrary*, *PE\_freeLibrary*

## PE\_loadLibrary

Loads a Portable Binary DLL into memory

### Declaration

```
PE_MODULE * WINAPI PE_loadLibrary(  
    const char *szDLLName,  
    ibool shared)
```

### Prototype In

drvlib/peloder.h

### Parameters

<i>szDLLName</i>	Name of the PE DLL library to load
<i>shared</i>	True to load module into shared memory

### Return Value

Handle to loaded PE DLL, or NULL on failure.

### Description

This function loads a Portable Binary DLL library from disk, relocates the code and returns a handle to the loaded library. This function will only work on DLL's that do not have any imports, since we don't resolve pimport dependencies in this function.

### See Also

*PE\_getProcAddress*, *PE\_freeLibrary*



## PE\_loadLibraryExt

Loads a Portable Binary DLL into memory from an open file

### Declaration

```
PE_MODULE * WINAPI PE_loadLibraryExt(
    FILE *f,
    ULONG startOffset,
    ULONG *size,
    BOOL shared)
```

### Prototype In

drvlib/peloder.h

### Parameters

<i>f</i>	Handle to open file to read driver from
<i>startOffset</i>	Offset to the start of the driver within the file
<i>size</i>	Place to store the size of the driver loaded
<i>shared</i>	True to load module into shared memory

### Return Value

Handle to loaded PE DLL, or NULL on failure.

### Description

This function loads a Portable Binary DLL library from disk, relocates the code and returns a handle to the loaded library. This function is the same as the regular *PE\_loadLibrary* except that it take a handle to an open file and an offset within that file for the DLL to load.

### See Also

*PE\_loadLibrary*, *PE\_getProcAddress*, *PE\_freeLibrary*

## PE\_loadLibraryMGL

Loads a Portable Binary DLL into memory

### Declaration

```
PE_MODULE * WINAPI PE_loadLibraryMGL(  
    const char *szDLLName,  
    ibool shared)
```

### Prototype In

drvlib/peloder.h

### Parameters

<i>szDLLName</i>	Name of the PE DLL library to load
<i>shared</i>	True to load module into shared memory

### Return Value

Handle to loaded PE DLL, or NULL on failure.

### Description

This function is the same as the regular *PE\_loadLibrary* function, except that it looks for the drivers in the MGL\_ROOT/drivers directory or a /drivers directory relative to the current directory.

### See Also

*PE\_loadLibraryMGL*, *PE\_getProcAddress*, *PE\_freeLibrary*

## REF2D\_loadDriver

Loads the 2D reference rasteriser device driver from disk.

### Declaration

```
ibool NAPI REF2D_loadDriver(
    GA_devCtx_FAR_ *hwCtx,
    N_int32 bitsPerPixel,
    N_int32 shared,
    REF2D_driver_FAR_ *_FAR_ *drv,
    PE_MODULE_FAR_ *_FAR_ *hModRef,
    ulong _FAR_ *size)
```

### Prototype In

snap/ref2d.h

### Parameters

<i>hwCtx</i>	Hardware context to load driver for (if any)
<i>bitsPerPixel</i>	Pixel depth for driver to load (1,4,8,16,24 or 32)
<i>shared</i>	True to load into shared memory
<i>drv</i>	Pointer to the place to store loaded driver address
<i>hModRef</i>	Pointer to the place to store the load driver module handle
<i>size</i>	Pointer to the place to store the size of the driver

### Return Value

True on success, false on failure.

### Description

This function loads a copy of the SNAP 2D Reference Rasteriser for the specific color depth from disk. This function will always load the SNAP reference rasteriser driver from the public MGL directories, and should not be used to load the reference rasteriser driver for use with a hardware device context. Rather you should use the *GA\_loadRef2d* function which automatically handles support for loading and initialising the Portait Display and Multi-Controller SNAP filter drivers.

However if you wish to load and use the SNAP reference rasteriser for drawing to system memory bitmaps, you can use this function instead.

### See Also

*GA\_loadRef2d*, *REF2D\_unloadDriver*

## REF2D\_queryFunctions

Returns the function pointers for the specified ref2d function group.

### Declaration

```
ibool NAPI REF2D_queryFunctions(
    REF2D_driver *ref2d,
    N_uint32 id,
    void _FAR_ *funcs)
```

### Prototype In

snap/graphics.h

### Parameters

<i>ref2d</i>	Ref2d driver to return the functions for
<i>id</i>	Identifier for the function group to get pointers for
<i>funcs</i>	Pointer to function block to fill in

### Return Value

True if the requested function group is available, false if not.

### Description

This function is the similar to *GA\_queryFunctions* function except that the functions are queried via the 2d reference rasteriser library. This is the function that application level code should use to get access to the SNAP rendering functions that are fleshed out with software rendered functions as necessary.

All application and device driver code must call this function and **never** call the *REF2D\_driver->QueryFunctions* member directly. This is important because in many cases special procedures must followed to allow ring 3 application code to directly call the function pointers returned by this function. See *GA\_queryFunctions* for more information.

---

**Note:** *To allow for future compatibility, all function blocks begin with a dwSize member. The caller is expected to fill in the dwSize member with the size of the function block being retrieved before calling REF2D\_queryFunctions. If the driver exports more functions than the application knows about, only a subset of the functions are copied to the application. If the application expects more functions than the driver provides, the non-existent functions are set to NULL pointers by REF2D\_queryFunctions, and the remainder copies from the driver.*

---

### See Also

*GA\_queryFunctions*

## REF2D\_unloadDriver

Unloads the 2D reference rasteriser from memory

### Declaration

```
void NAPI REF2D_unloadDriver(  
    REF2D_driver_FAR_ *drv,  
    PE_MODULE_FAR_ *hModule)
```

### Prototype In

snap/ref2d.h

### Parameters

<i>drv</i>	Pointer to driver to unload
<i>hModule</i>	Pointer to the module for the driver to unload

### Description

This function unloads the SNAP 2D Reference Rasteriser from memory. This is used for when a reference rasteriser is loaded from disk independantly of the hardware drivers, for use in rendering to system memory buffers completely in software.

### See Also

*REF2D\_loadDriver*

## *Type Definitions*

---

## DDC\_ChannelsType

### Declaration

```
typedef enum {  
    SCI_channelMonitorPrimary    = 0x0000,  
    SCI_channelMonitorSecondary = 0x0001,  
    SCI_channelTVTuner          = 0x0100  
} DDC_ChannelsType
```

### Prototype In

snap/ddc.h

### Description

Flags passed to the Serial Control Interface functions to determine what I2C communications channel should be used. Usually the CRT monitor channel is used, but you can use different channels to control different devices.

### Members

<i>SCI_channelMonitorPrimary</i>	Primary head display monitor channel
<i>SCI_channelMonitorSecondary</i>	Secondary head display monitor channel
<i>SCI_channelTVTuner</i>	TV-Tuner channel

## DDC\_DPMSStatesType

### Declaration

```
typedef enum {
    DPMS_on           = 0,
    DPMS_standby      = 1,
    DPMS_suspend      = 2,
    DPMS_off          = 4
} DDC_DPMSStatesType
```

### Prototype In

snap/ddc.h

### Description

DPMS state values to pass to the DPMSsetState device driver call. This enumeration defines the values to set a specific power down state, and are based on the values defined in the VESA DPMS 1.0 specification. Please consult this or later versions of this specification for more information.

### Members

<i>DPMS_on</i>	Return the controller to the ON state
<i>DPMS_standby</i>	Set the controller to the Stand-By power down state
<i>DPMS_suspend</i>	Set the controller to the Suspend power down state
<i>DPMS_off</i>	Set the controller to the Off power down state



## DDC\_SCIFlagsType

### Declaration

```
typedef enum {
    SCI_writeSCL      = 0x01,
    SCI_writeSDA      = 0x02,
    SCI_readSCL       = 0x04,
    SCI_readSDA       = 0x08,
    SCI_blankFlag     = 0x10
} DDC_SCIFlagsType
```

### Prototype In

snap/ddc.h

### Description

Flags for the level of Serial Control Interface supported by the hardware and returned by the SCIdetect device driver call. Generally, application software will not control the SCI interface directly, but will use the higher level DDC and MCS functions, which implement packet based protocols over the SCI interface.

### Members

<i>SCI_writeSCL</i>	Writing the SCL Clock Line is supported
<i>SCI_writeSDA</i>	Writing the SDA Data Line is supported
<i>SCI_readSCL</i>	Reading the SCL Clock Line is supported
<i>SCI_readSDA</i>	Reading the SDA Data Line is supported
<i>SCI_blankFlag</i>	Screen will be blanked during communications

## DDC\_errCode

### Declaration

```
typedef enum {
    ddcOk                = 0,
    ddcNotAvailable      = 1,
    ddcNoCommunication   = -1
} DDC_errCode
```

### Prototype In

snap/ddc.h

### Description

Returns values from *DDC\_initExt* and *MCS\_initExt* functions.

The *ddcOk* value indicates that the DDC communications channel was initialised successfully for both the graphics card and the monitor.

The *ddcNotAvailable* value indicates that the graphics card does not support DDC communications or that this feature is disabled for the device driver.

The *ddcNoCommunication* value indicates that the graphics card does support DDC communications, however the monitor is not responding on the DDC communications channel. The most likely cause of this is that the monitor attached to the graphics card is not DDC2B enabled.

### Members

<i>ddcOk</i>	DDC communications initialised correctly
<i>ddcNotAvailable</i>	DDC is not available
<i>ddcNoCommunication</i>	DDC is available but not communicating

## EDID\_detailedTiming

### Declaration

```
typedef struct {
    ushort  pixelClock;
    ushort  hActive;
    ushort  hBlank;
    ushort  hSyncOffset;
    ushort  hSyncWidth;
    ushort  hBorder;
    ushort  vActive;
    ushort  vBlank;
    ushort  vSyncOffset;
    ushort  vSyncWidth;
    ushort  vBorder;
    ushort  hSize;
    ushort  vSize;
    char     hSyncPol;
    char     vSyncPol;
    ushort  hFreq;
    uchar    Hz;
} EDID_detailedTiming
```

### Prototype In

snap/ddc.h

### Description

Structure to describe all detailed timings

### Members

<i>pixelClock</i>	Pixel clock in Hz / 10,000
<i>hActive</i>	Horizontal active display value (X resolution)
<i>hBlank</i>	Horizontal blank start position
<i>hSyncOffset</i>	Horizontal sync offset from blank start
<i>hSyncWidth</i>	Horizontal sync width
<i>hBorder</i>	Horizontal border width
<i>vActive</i>	Vertical active display (Y resolution)
<i>vBlank</i>	Vertical blank start position
<i>vSyncOffset</i>	Vertical sync offset from blank start
<i>vSyncWidth</i>	Vertical sync width
<i>vBorder</i>	Vertical border width
<i>hSize</i>	Horizontal image size in mm
<i>vSize</i>	Vertical image size in mm
<i>hSyncPol</i>	Horizontal sync polarity ('+' or '-')
<i>vSyncPol</i>	Vertical sync polarity ('+' or '-')
<i>hFreq</i>	Horizontal frequency in KHz * 100
<i>Hz</i>	Vertical frequency in Hz

## EDID\_displayTypes

### Declaration

```
typedef enum {  
    EDID_GrayScale,  
    EDID_RGBColor,  
    EDID_NonRGBColor  
} EDID_displayTypes
```

### Prototype In

snap/ddc.h

### Description

This enumeration defines the monitor display types stored in the displayType field of the *EDID\_record* structure.

### Members

<i>EDID_GrayScale</i>	Monochrome/Grayscale monitor
<i>EDID_RGBColor</i>	RGB color monitor
<i>EDID_NonRGBColor</i>	NonRGB color monitor

## EDID\_flags

### Declaration

```
typedef enum {
    EDID_DPMSStandBy      = 0x0001,
    EDID_DPMSSuspend      = 0x0002,
    EDID_DPMSSOff         = 0x0004,
    EDID_DPMSEnabled      = 0x0007,
    EDID_GTFEnabled       = 0x0008,
    EDID_DDC2AB           = 0x0010,
    EDID_Blank2Blank      = 0x0020,
    EDID_SyncSeparate     = 0x0040,
    EDID_SyncComposite    = 0x0080,
    EDID_SyncOnGreen      = 0x0100,
    EDID_NeedSerration    = 0x0200,
    EDID_PreferredTiming  = 0x0400
} EDID_flags
```

### Prototype In

snap/ddc.h

### Description

This enumeration defines the values stored in the flags field of the *EDID\_record* structure.

### Members

<i>EDID_DPMSStandBy</i>	The DPMS Standby state is supported
<i>EDID_DPMSSuspend</i>	The DPMS Suspend state is supported
<i>EDID_DPMSSOff</i>	The DPMS Off state is supported
<i>EDID_DPMSEnabled</i>	Monitor supports DPMS Power Management
<i>EDID_GTFEnabled</i>	Monitor supports GTF timings
<i>EDID_DDC2AB</i>	Monitor supports DDC2AB interface
<i>EDID_Blank2Blank</i>	Monitor requires Blank-to-Blank setup
<i>EDID_SyncSeparate</i>	Monitor supports separate syncs
<i>EDID_SyncComposite</i>	Monitor supports composite syncs
<i>EDID_SyncOnGreen</i>	Monitor supports Sync on Green
<i>EDID_NeedSerration</i>	VSynC serration is required
<i>EDID_PreferredTiming</i>	Detailed timing 1 is preferred timing for monitor

## EDID\_maxResCodes

### Declaration

```
typedef enum {
    MaxRes_640x480,
    MaxRes_800x600,
    MaxRes_1024x768,
    MaxRes_1152x864,
    MaxRes_1280x1024,
    MaxRes_1600x1200,
    MaxRes_1800x1350,
    MaxRes_1920x1440,
    MaxRes_2048x1536
} EDID_maxResCodes
```

### Prototype In

snap/ddc.h

### Description

This enumeration defines the list of maximum resolutions as reported in the *EDID\_record* structure. Note that these values determine if the monitor can handle the specific resolution in the 60Hz non-interlaced format. These values may be less than what you would expect if the monitor can handle 1024x768 interlaced, but not in non-interlaced mode.

### Members

<i>MaxRes_640x480</i>	Maximum resolution is 640x480 @ 60Hz NI
<i>MaxRes_800x600</i>	Maximum resolution is 800x600 @ 60Hz NI
<i>MaxRes_1024x768</i>	Maximum resolution is 1024x768 @ 60Hz NI
<i>MaxRes_1152x864</i>	Maximum resolution is 1152x864 @ 60Hz NI
<i>MaxRes_1280x1024</i>	Maximum resolution is 1280x1024 @ 60Hz NI
<i>MaxRes_1600x1200</i>	Maximum resolution is 1600x1200 @ 60Hz NI
<i>MaxRes_1800x1350</i>	Maximum resolution is 1800x1350 @ 60Hz NI
<i>MaxRes_1920x1440</i>	Maximum resolution is 1920x1440 @ 60Hz NI
<i>MaxRes_2048x1536</i>	Maximum resolution is 2048x1536 @ 60Hz NI

## EDID\_record

### Declaration

```
typedef struct {
    ushort    version;
    char      mfrID[4];
    char      mfrName[40];
    char      modelName[40];
    char      serialNo[14];
    char      PNPID[8];
    ushort    productID;
    ulong     serialID;
    uchar     mfrWeek;
    ushort    mfrYear;
    uchar     signalLevel;
    uchar     displayType;
    uchar     maxResolution;
    uchar     minHScan;
    uchar     maxHScan;
    uchar     minVScan;
    uchar     maxVScan;
    ushort    maxPCLK;
    ushort    flags;
    uchar     maxHSize;
    uchar     maxVSize;
    N_fix32   gamma;
    N_fix32   Rx,Ry;
    N_fix32   Gx,Gy;
    N_fix32   Bx,By;
    N_fix32   Wx,Wy;
    uchar     numStandardTimings;
    uchar     numDetailedTimings;
    EDID_standardTiming standardTimings[MAX_STANDARD_TIMINGS];
    EDID_detailedTiming detailedTimings[MAX_DETAILED_TIMINGS];
} EDID_record
```

### Prototype In

snap/ddc.h

### Description

Main structure containing the information parsed from the binary EDID data returned from the monitor.

### Members

<i>version</i>	EDID version Number (in BCD)
<i>mfrID</i>	3 byte EISA manufacturer ID
<i>mfrName</i>	ASCII manufacturer name (Unknown if not found)
<i>modelName</i>	ASCII model name for monitor (Unknown if not found)
<i>serialNo</i>	ASCII serial number (Unknown if not found)
<i>PNPID</i>	8 character Plug and Play ID
<i>productID</i>	16-bit product ID code
<i>serialID</i>	32-bit product serial number
<i>mfrWeek</i>	Week of manufacture (0-52)
<i>mfrYear</i>	Year of manufacture

<i>signalLevel</i>	Signal level code ( <i>EDID_signalLevels</i> )
<i>displayType</i>	Display type code ( <i>EDID_displayTypes</i> )
<i>maxResolution</i>	Maximum resolution ID ( <i>EDID_maxResCodes</i> )
<i>minHScan</i>	Minimum horizontal scan (kHz)
<i>maxHScan</i>	Maximum horizontal scan (kHz)
<i>minVScan</i>	Minimum vertical scan (Hz)
<i>maxVScan</i>	Maximum vertical scan (Hz)
<i>maxPCLK</i>	Maximum pixel clock (MHz)
<i>flags</i>	Capabilities flags ( <i>EDID_flags</i> )
<i>maxHSize</i>	Maximum horizontal size (cm)
<i>maxVSize</i>	Maximum vertical size (cm)
<i>gamma</i>	Display transfer characteristic (16.16 fixed point)
<i>Rx</i>	Red X chromaticity characteristic (16.16 fixed point)
<i>Ry</i>	Red Y chromaticity characteristic (16.16 fixed point)
<i>Gx</i>	Green X chromaticity characteristic (16.16 fixed point)
<i>Gy</i>	Green Y chromaticity characteristic (16.16 fixed point)
<i>Bx</i>	Blue X chromaticity characteristic (16.16 fixed point)
<i>By</i>	Blue Y chromaticity characteristic (16.16 fixed point)
<i>Wx</i>	Default white point X characteristic (16.16 fixed point)
<i>Wy</i>	Default white point Y characteristic (16.16 fixed point)
<i>numStandardTimings</i>	Number of standard timings listed
<i>numDetailedTimings</i>	Number of detailed timings listed
<i>standardTimings</i>	List of standard timings
<i>detailedTimings</i>	List of detailed timings



## EDID\_signalLevels

### Declaration

```
typedef enum {
    EDID_Level_0700_0300_10P,
    EDID_Level_0714_0286_10P,
    EDID_Level_1000_0400_14P,
    EDID_Level_0700_0300_07P,
    EDID_Level_Digital
} EDID_signalLevels
```

### Prototype In

snap/ddc.h

### Description

This enumeration defines the signal level types stored in the signalLevel field of the *EDID\_record* structure.

### Members

<i>EDID_Level_0700_0300_10P</i>	Analog 0.700 - 0.300 (1.0V p-p)
<i>EDID_Level_0714_0286_10P</i>	Analog 0.714 - 0.286 (1.0V p-p)
<i>EDID_Level_1000_0400_14P</i>	Analog 1.000 - 0.400 (1.4V p-p)
<i>EDID_Level_0700_0300_07P</i>	Analog 0.700 - 0.300 (0.7V p-p)
<i>EDID_Level_Digital</i>	Digital signal

## EDID\_standardTiming

### Declaration

```
typedef struct {  
    ushort  xRes;  
    ushort  yRes;  
    uchar   Hz;  
    uchar   flags;  
} EDID_standardTiming
```

### Prototype In

snap/ddc.h

### Description

Structure to describe all established and standard timings

### Members

<i>xRes</i>	Horizontal resolution in pixels
<i>yRes</i>	Vertical resolution in lines
<i>Hz</i>	Vertical refresh rate in Hz
<i>flags</i>	Flags ( <i>EDID_timingTypes</i> )

## EDID\_timingTypes

### Declaration

```
typedef enum {  
    EDID_VGACompatible,  
    EDID_XGACompatible,  
    EDID_MacCompatible,  
    EDID_VESASStandard,  
    EDID_MaxTimingType  
} EDID_timingTypes
```

### Prototype In

snap/ddc.h

### Description

This enumeration defines the values stored in the flags field of the *EDID\_standardTiming* structure.

### Members

<i>EDID_VGACompatible</i>	VGA compatible timing
<i>EDID_XGACompatible</i>	XGA compatible timing
<i>EDID_MacCompatible</i>	Macintosh compatible timing
<i>EDID_VESASStandard</i>	VESA standard timing

## GA\_2DRenderFuncs

**Prototype In**

snap/graphics.h

**Description**

Function group containing all the device driver functions related to managing drawing in the framebuffer using the 2D graphics accelerator. This group of functions does not contain any functions related to state management, just drawing.

Generally applications or shell drivers should request this block of functions from the 2d reference rasteriser library, not directly from the graphics accelerator. This will allow the library to fill in all rendering functions with software rendering as necessary automatically.

---

**Note:** *Be sure to fill in the dwSize member of this structure when you call GA\_queryFunctions to the correct size of the structure at compile time!*

---

## BitBlt

Copy a block of video memory to another location in video memory.

### Declaration

```
void NAPI_GA_2DRenderFuncs::BitBlt(
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 width,
    N_int32 height,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 mix)
```

### Prototype In

snap/graphics.h

### Parameters

<i>srcLeft</i>	Left coordinate of the source rectangle to copy
<i>srcTop</i>	Top coordinate of the source rectangle to copy
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>mix</i>	Mix code for the copy ( <i>GA_mixCodesType</i> )

### Description

This function copies a rectangular region of video memory from one location to another. This routine will copy a rectangular region of video memory from (srcLeft, srcTop, srcLeft+width-1, srcTop+height-1) to (dstLeft, dstTop) within video memory with the specified mix, and will also correctly handle cases of overlapping regions in video memory. The mix code will be used to combine the source bitmap data with the pixels in the destination bitmap.

### See Also

*BitBltLin*, *BitBltSys*, *BitBltBM*, *SrcTransBlt*, *DstTransBlt*, *BitBltFx*

## BitBltBM

Copy a block of system memory to a location in video memory with Bus Mastering.

### Declaration

```
void NAPI_GA_2DRenderFuncs::BitBltBM(
    void *srcAddr,
    N_int32 srcPhysAddr,
    N_int32 srcPitch,
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 width,
    N_int32 height,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 mix)
```

### Prototype In

snap/graphics.h

### Parameters

<i>srcAddr</i>	Address of source bitmap in system memory
<i>srcPhysAddr</i>	Physical address of source bitmap in system memory
<i>srcPitch</i>	Pitch of source bitmap in bytes
<i>srcLeft</i>	Left coordinate within source bitmap to copy
<i>srcTop</i>	Top coordinate within source bitmap to copy
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>mix</i>	Mix code for the copy ( <i>GA_mixCodesType</i> )

### Description

This routine will copy a bitmap from system memory with a physical starting address of *srcPhysAddr* to the destination rectangle (*dstLeft*, *dstTop*, *dstLeft*+*width*-1, *dstTop*+*height*-1) with the specified mix. The mix code will be used to combine the source bitmap data with the pixels in the destination bitmap. The *srcPhysAddr* value points to the start of the bitmap data in system memory as a *physical* memory address, not a linear memory address that the application software normally deals with. It is up to the calling application to use the necessary OS services to allocate a block of contiguous physical memory for the bitmap data, and to obtain the physical memory address to be passed into this function.

This version is different to the *BitBltSys* function in that the bitmap data is copied using Bus Mastering by the graphics accelerator, which allows this function to return before the copy has completed and the accelerator will complete the copy in the background with a DMA Bus Master operation. If this hardware supports Bus Mastering and this function is available, it will usually be the fastest method to copy a block of system memory to video memory.

Note that the `srcLeft` and `srcTop` coordinates define an offset within the source bitmap to be copied, so it will copy only a portion of the memory bitmap.

**See Also**

*BitBlt, BitBltLin, BitBltSys, SrcTransBlt, DstTransBlt, BitBltEx*

## BitBltColorPatt

Copy a block of video memory, with a color pattern applied

### Declaration

```
void NAPI GA_2DRenderFuncs::BitBltColorPatt(
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 width,
    N_int32 height,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 rop3)
```

### Prototype In

snap/graphics.h

### Parameters

<i>srcLeft</i>	Left coordinate of the source rectangle to copy
<i>srcTop</i>	Top coordinate of the source rectangle to copy
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>rop3</i>	Microsoft ROP3 code for the copy ( <i>GA_rop3CodesType</i> )

### Description

This function is identical to the regular *BitBlt* function, except that it also applies the currently active 8x8 color pattern to the destination. The source data, pattern data and destination data are combined together according to the value passed in the *rop3* parameter.

### See Also

*BitBltColorPattLin*, *BitBltColorPattSys*, *BitBltColorPattBM*, *SrcTransBlt*, *DstTransBlt*



## BitBltColorPattBM

Copy a block of system memory to video memory with Bus Mastering, with a color pattern applied

### Declaration

```
void NAPI_GA_2DRenderFuncs::BitBltColorPattBM(
    void *srcAddr,
    N_int32 srcPhysAddr,
    N_int32 srcPitch,
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 width,
    N_int32 height,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 rop3)
```

### Prototype In

snap/graphics.h

### Parameters

<i>srcAddr</i>	Address of source bitmap in system memory
<i>srcPhysAddr</i>	Physical address of source bitmap in system memory
<i>srcPitch</i>	Pitch of source bitmap in bytes
<i>srcLeft</i>	Left coordinate within source bitmap to copy
<i>srcTop</i>	Top coordinate within source bitmap to copy
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>rop3</i>	Microsoft ROP3 code for the copy ( <i>GA_rop3CodesType</i> )

### Description

This function is identical to the regular *BitBltSys* function, except that it also applies the currently active 8x8 color pattern to the destination. The source data, pattern data and destination data are combined together according to the value passed in the *rop3* parameter.

### See Also

*BitBltColorPatt*, *BitBltColorPattLin*, *BitBltColorPattSys*, *SrcTransBlt*, *DstTransBlt*

## BitBltColorPattLin

Copy a linear block of video memory, with a color pattern applied

### Declaration

```
void NAPI_GA_2DRenderFuncs::BitBltColorPattLin(
    N_int32 srcOfs,
    N_int32 srcPitch,
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 width,
    N_int32 height,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 rop3)
```

### Prototype In

snap/graphics.h

### Parameters

<i>srcOfs</i>	Offset of source bitmap in video memory
<i>srcPitch</i>	Pitch of source bitmap in bytes
<i>srcLeft</i>	Left coordinate within source bitmap to copy
<i>srcTop</i>	Top coordinate within source bitmap to copy
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>rop3</i>	Microsoft ROP3 code for the copy ( <i>GA_rop3CodesType</i> )

### Description

This function is identical to the regular *BitBltLin* function, except that it also applies the currently active 8x8 color pattern to the destination. The source data, pattern data and destination data are combined together according to the value passed in the *rop3* parameter.

### See Also

*BitBltColorPatt*, *BitBltColorPattSys*, *BitBltColorPattBM*, *SrcTransBlt*, *DstTransBlt*

## BitBltColorPattSys

Copy a block of system memory to video memory, with a color pattern applied

### Declaration

```
void NAPI_GA_2DRenderFuncs::BitBltColorPattSys(
    void *srcAddr,
    N_int32 srcPitch,
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 width,
    N_int32 height,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 rop3,
    N_int32 flipY)
```

### Prototype In

snap/graphics.h

### Parameters

<i>srcAddr</i>	Address of source bitmap in system memory
<i>srcPitch</i>	Pitch of source bitmap in bytes
<i>srcLeft</i>	Left coordinate within source bitmap to copy
<i>srcTop</i>	Top coordinate within source bitmap to copy
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>rop3</i>	Microsoft ROP3 code for the copy ( <i>GA_rop3CodesType</i> )
<i>flipY</i>	True if the image should be flipped vertically

### Description

This function is identical to the regular *BitBltSys* function, except that it also applies the currently active 8x8 color pattern to the destination. The source data, pattern data and destination data are combined together according to the value passed in the rop3 parameter.

### See Also

*BitBltColorPatt*, *BitBltColorPattSys*, *BitBltColorPattBM*, *SrcTransBlt*, *DstTransBlt*

## BitBltFx

Copy a block of video memory to another location in video memory with optional effects.

### Declaration

```
void NAPI_GA_2DRenderFuncs::BitBltFx(
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 srcWidth,
    N_int32 srcHeight,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 dstWidth,
    N_int32 dstHeight,
    GA_bltFx *fx)
```

### Prototype In

snap/graphics.h

### Parameters

<i>srcLeft</i>	Left coordinate of the source rectangle to copy
<i>srcTop</i>	Top coordinate of the source rectangle to copy
<i>srcWidth</i>	Width of the source rectangle in pixels
<i>srcHeight</i>	Height of the source rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>dstWidth</i>	Width of the destination rectangle in pixels
<i>dstHeight</i>	Height of the destination rectangle in scanlines
<i>fx</i>	<i>GA_bltFx</i> structure describing the requested effects

### Description

This function copies a rectangular region of video memory from one location to another with the optional effects described in the *GA\_bltFx* structure. Currently this function can perform stretching, source and destination transparency and flipping depending on what features the underlying hardware supports, with an optional mix code. This routine will copy the rectangular region of video memory from (*srcLeft*, *srcTop*, *srcLeft*+*srcWidth*-1, *srcTop*+*srcHeight*-1) to (*dstLeft*, *dstTop*, *dstLeft*+*dstWidth*-1, *dstTop*+*dstHeight*-1) within video memory. Note that the source and destination rectangle dimensions may be different in, which is the case for doing a copy with bitmap stretching. If the *GA\_bltFx* structure does not indicate stretching is in effect, the *dstHeight* and *dstWidth* parameters will be ignored and only the *srcWidth* and *srcHeight* parameters will be used. The results of this routine are undefined if the video memory regions overlap.

---

**Note:** *Some of the features may not be supported at the same time, and it is up to the application programmer to call the *BitBltFxTest* function to determine what features are supported before calling this function. Calling this function with an unsupported set of features will result in undefined behaviour.*

---

**See Also**

*BitBltFxTest, BitBltFxLin, BitBltFxSys, BitBltFxBM, SrcTransBlt, DstTransBlt, BitBlt*

## BitBltFxBM

Copy a linear block of video memory to another location in video memory with optional effects and bus mastering.

### Declaration

```
void NAPI GA_2DRenderFuncs::BitBltFxBM(
    void *srcAddr,
    N_int32 srcPhysAddr,
    N_int32 srcPitch,
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 srcWidth,
    N_int32 srcHeight,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 dstWidth,
    N_int32 dstHeight,
    GA_bltFx *fx)
```

### Prototype In

snap/graphics.h

### Parameters

<i>srcAddr</i>	Address of source bitmap in system memory
<i>srcPhysAddr</i>	Physical address of source bitmap in system memory
<i>srcPitch</i>	Pitch of source bitmap in bytes
<i>srcLeft</i>	Left coordinate of the source rectangle to copy
<i>srcTop</i>	Top coordinate of the source rectangle to copy
<i>srcWidth</i>	Width of the source rectangle in pixels
<i>srcHeight</i>	Height of the source rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>dstWidth</i>	Width of the destination rectangle in pixels
<i>dstHeight</i>	Height of the destination rectangle in scanlines
<i>fx</i>	<i>GA_bltFx</i> structure describing the requested effects

### Description

This function copies a bitmap from system memory with a physical starting address of *srcPhysAddr* to video memory with the optional features described in the *GA\_bltFx* structure. Currently this function can perform stretching, source and destination transparency and flipping depending on what features the underlying hardware supports, with an optional mix code. This routine will copy the region (*srcLeft*, *srcTop*, *srcLeft*+*srcWidth*-1, *srcTop*+*srcHeight*-1) from the source bitmap to (*dstLeft*, *dstTop*, *dstLeft*+*dstWidth*-1, *dstTop*+*dstHeight*-1) in video memory. The *srcPhysAddr* value points to the start of the bitmap data in system memory as a *physical* memory address, not a linear memory address that the application software normally deals with. It is up to the calling application to use the necessary OS services to allocate a block of contiguous physical memory for the bitmap data, and to obtain the physical memory address to be passed into this function. Note that the source and destination rectangle

dimensions may be different in, which is the case for doing a copy with bitmap stretching. If the *GA\_bltFx* structure does not indicate stretching is in effect, the *dstHeight* and *dstWidth* parameters will be ignored and only the *srcWidth* and *srcHeight* parameters will be used.

This version is different to the *BitBltFxSys* function in that the bitmap data is copied using Bus Mastering by the graphics accelerator, which allows this function to return before the copy has completed and the accelerator will complete the copy in the background with a DMA Bus Master operation. If this hardware supports Bus Mastering and this function is available, it will usually be the fastest method to copy a block of system memory to video memory.

Note that the *srcLeft* and *srcTop* coordinates define an offset within the source bitmap to be copied, so it will copy only a portion of the memory bitmap.

---

**Note:** *Some of the features may not be supported at the same time, and it is up to the application programmer to call the *BitBltFxTest* function to determine what features are supported before calling this function. Calling this function with an unsupported set of features will result in undefined behaviour.*

---

#### **See Also**

*BitBltFxTest, BitBltFx, BitBltFxLin, BitBltFxBM, SrcTransBlt, DstTransBlt, BitBlt*

## BitBltFxLin

Copy a linear block of video memory to another location in video memory with optional effects.

### Declaration

```
void NAPI GA_2DRenderFuncs::BitBltFxLin(
    N_int32 srcOfs,
    N_int32 srcPitch,
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 srcWidth,
    N_int32 srcHeight,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 dstWidth,
    N_int32 dstHeight,
    GA_bltFx *fx)
```

### Prototype In

snap/graphics.h

### Parameters

<i>srcOfs</i>	Offset of source bitmap in video memory
<i>srcPitch</i>	Pitch of source bitmap in bytes
<i>srcLeft</i>	Left coordinate of the source rectangle to copy
<i>srcTop</i>	Top coordinate of the source rectangle to copy
<i>srcWidth</i>	Width of the source rectangle in pixels
<i>srcHeight</i>	Height of the source rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>dstWidth</i>	Width of the destination rectangle in pixels
<i>dstHeight</i>	Height of the destination rectangle in scanlines
<i>fx</i>	<i>GA_bltFx</i> structure describing the requested effects

### Description

This function copies a linear region of video memory from one location to another with the optional features described in the *GA\_bltFx* structure. Note that the value of *srcOfs* must be aligned to the boundary specified in the *BitmapStartAlign* member of the *GA\_modeInfo* structure, and the *srcPitch* value must be padded to multiples of the *BitmapStridePad* member of the *GA\_modeInfo* structure. Currently this function can perform stretching, source and destination transparency and flipping depending on what features the underlying hardware supports, with an optional mix code. This routine will copy the region (*srcLeft*, *srcTop*, *srcLeft*+*srcWidth*-1, *srcTop*+*srcHeight*-1) from the source bitmap to (*dstLeft*, *dstTop*, *dstLeft*+*dstWidth*-1, *dstTop*+*dstHeight*-1) in video memory. Note that the source and destination rectangle dimensions may be different in, which is the case for doing a copy with bitmap stretching. If the *GA\_bltFx* structure does not indicate stretching is in effect, the *dstHeight* and *dstWidth* parameters will be ignored and only the *srcWidth* and *srcHeight* parameters will be used. The results of this routine are undefined if the video memory regions overlap.



Note that the `srcLeft` and `srcTop` coordinates define an offset within the source bitmap to be copied, so it will copy only a portion of an offscreen bitmap. This is useful for storing multiple images in a single offscreen bitmap, or for handling the case of software clipping offscreen bitmaps if the destination lies outside of the software clip rectangle for the destination buffer.

This version is different to the standard *BitBltEx* function in that the source bitmap to be copied can be non-conforming, and can have a different logical scanline width to the destination bitmap. This allows the bitmaps to be stored contiguously in offscreen video memory, rather than requiring the offscreen video memory to be divided up into rectangular regions, resulting in more efficient use of available offscreen memory for bitmap storage.

---

**Note:** *The value of `srcOfs` must be aligned to the boundary specified in the `BitmapStartAlign` member of the `GA_modeInfo` structure, and the `dstPitch` value must be padded to multiples of the `BitmapStridePad` member of the `GA_modeInfo` structure.*

**Note:** *Some of the features may not be supported at the same time, and it is up to the application programmer to call the `BitBltExTest` function to determine what features are supported before calling this function. Calling this function with an unsupported set of features will result in undefined behaviour.*

---

#### **See Also**

*BitBltExTest, BitBltEx, BitBltExSys, BitBltExBM, SrcTransBlt, DstTransBlt, BitBlt*

## BitBltFxSys

Copy a linear block of video memory to another location in video memory with optional effects.

### Declaration

```
void NAPI GA_2DRenderFuncs::BitBltFxSys(
    void *srcAddr,
    N_int32 srcPitch,
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 srcWidth,
    N_int32 srcHeight,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 dstWidth,
    N_int32 dstHeight,
    GA_bltFx *fx)
```

### Prototype In

snap/graphics.h

### Parameters

<i>srcAddr</i>	Address of source bitmap in system memory
<i>srcPitch</i>	Pitch of source bitmap in bytes
<i>srcLeft</i>	Left coordinate of the source rectangle to copy
<i>srcTop</i>	Top coordinate of the source rectangle to copy
<i>srcWidth</i>	Width of the source rectangle in pixels
<i>srcHeight</i>	Height of the source rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>dstWidth</i>	Width of the destination rectangle in pixels
<i>dstHeight</i>	Height of the destination rectangle in scanlines
<i>fx</i>	<i>GA_bltFx</i> structure describing the requested effects

### Description

This function copies a bitmap from system memory with a starting address of *srcAddr* to video memory with the optional features described in the *GA\_bltFx* structure. Currently this function can perform stretching, source and destination transparency and flipping depending on what features the underlying hardware supports, with an optional mix code. This routine will copy the region (*srcLeft*, *srcTop*, *srcLeft*+*srcWidth*-1, *srcTop*+*srcHeight*-1) from the source bitmap to (*dstLeft*, *dstTop*, *dstLeft*+*dstWidth*-1, *dstTop*+*dstHeight*-1) in video memory. Note that the source and destination rectangle dimensions may be different in, which is the case for doing a copy with bitmap stretching. If the *GA\_bltFx* structure does not indicate stretching is in effect, the *dstHeight* and *dstWidth* parameters will be ignored and only the *srcWidth* and *srcHeight* parameters will be used.

Note that the *srcLeft* and *srcTop* coordinates define an offset within the source bitmap to be copied, so it will copy only a portion of an offscreen bitmap. This is useful for storing

multiple images in a single offscreen bitmap, or for handling the case of software clipping offscreen bitmaps if the destination lies outside of the software clip rectangle for the destination buffer.

---

**Note:** *Some of the features may not be supported at the same time, and it is up to the application programmer to call the `BitBltFxTest` function to determine what features are supported before calling this function. Calling this function with an unsupported set of features will result in undefined behaviour.*

---

**See Also**

*`BitBltFxTest`, `BitBltFx`, `BitBltFxLin`, `BitBltFxBM`, `SrcTransBlt`, `DstTransBlt`, `BitBlt`*

## BitBltFxTest

Tests if a set of *BitBltFx* features are supported.

### Declaration

```
N_int32 NAPI GA_2DRenderFuncs::BitBltFxTest(
    GA_bltFx *fx)
```

### Prototype In

snap/graphics.h

### Parameters

*fx*                      *GA\_bltFx* structure to check support for

### Return Value

1 if the features are supported, 0 if not.

### Description

This function allows an application to fill in an *GA\_bltFx* structure with the desired capabilities and to query the driver to determine if those capabilities are available. The driver reports the supported *BitBltFx* capabilities via the flags and values returned in the *BitBltCaps* field of the *GA\_modeInfo* structure. However some of the features reported as being available may be mutually exclusive of each other, for instance the hardware may support stretching and source transparency, but not at the same time. This function allows an application program to determine this at runtime, by requesting both stretching and transparency (for instance *afBltStretchNearest* | *afBltColorKeySrcSingle*) in the *GA\_bltFx* structure and calling this function to determine if the hardware supports these features together. If the hardware can perform a *BitBltFx* function with the specified *GA\_bltFx* features, this function will return a value of 1. If not it will return a value of 0.

### See Also

*BitBltFx*, *BitBltFxLin*, *BitBltFxSys*, *BitBltFxBM*

## BitBltLin

Copy a linear block of video memory to another location in video memory.

### Declaration

```
void NAPI_GA_2DRenderFuncs::BitBltLin(
    N_int32 srcOfs,
    N_int32 srcPitch,
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 width,
    N_int32 height,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 mix)
```

### Prototype In

snap/graphics.h

### Parameters

<i>srcOfs</i>	Offset of source bitmap in video memory
<i>srcPitch</i>	Pitch of source bitmap in bytes
<i>srcLeft</i>	Left coordinate within source bitmap to copy
<i>srcTop</i>	Top coordinate within source bitmap to copy
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>mix</i>	Mix code for the copy ( <i>GA_mixCodesType</i> )

### Description

This routine will copy a linear region of video memory from *srcOfs* from the start of video memory to the destination rectangle (*dstLeft*, *dstTop*, *dstLeft*+*width*-1, *dstTop*+*height*-1) with the specified mix. The mix code will be used to combine the source bitmap data with the pixels in the destination bitmap. The results of this routine are undefined if the video memory regions overlap.

Note that the *srcLeft* and *srcTop* coordinates define an offset within the source bitmap to be copied, so it is possible to only copy a portion of an offscreen bitmap. This is useful for storing multiple images in a single offscreen bitmap, or for handling the case of software clipping offscreen bitmaps if the destination lies outside of the software clip rectangle for the destination buffer.

This version is different to the standard *BitBlt* function in that the source bitmap to be copied can be non-conforming, and can have a different logical scanline width to the destination bitmap. This allows the bitmaps to be stored contiguously in offscreen video memory, rather than requiring the offscreen video memory to be divided up into rectangular regions, resulting in more efficient use of available offscreen memory for bitmap storage.

---

**Note:** *The value of `srcOfs` must be aligned to the boundary specified in the `BitmapStartAlign` member of the `GA_modeInfo` structure, and the `srcPitch` value must be padded to multiples of the `BitmapStridePad` member of the `GA_modeInfo` structure.*

---

**See Also**

*BitBlt, BitBltSys, BitBltBM, SrcTransBlt, DstTransBlt, BitBltFx*

## BitBltPatt

Copy a block of video memory, with a mono pattern applied

### Declaration

```
void NAPI GA_2DRenderFuncs::BitBltPatt(
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 width,
    N_int32 height,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 rop3)
```

### Prototype In

snap/graphics.h

### Parameters

<i>srcLeft</i>	Left coordinate of the source rectangle to copy
<i>srcTop</i>	Top coordinate of the source rectangle to copy
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>rop3</i>	Microsoft ROP3 code for the copy ( <i>GA_rop3CodesType</i> )

### Description

This function is identical to the regular *BitBlt* function, except that it also applies the currently active 8x8 monochrome pattern to the destination. The source data, pattern data and destination data are combined together according to the value passed in the *rop3* parameter.

### See Also

*BitBltPattLin*, *BitBltPattSys*, *BitBltPattBM*, *SrcTransBlt*, *DstTransBlt*

## BitBltPattBM

Copy a block of system memory to video memory with Bus Mastering, with a mono pattern applied

### Declaration

```
void NAPI_GA_2DRenderFuncs::BitBltPattBM(
    void *srcAddr,
    N_int32 srcPhysAddr,
    N_int32 srcPitch,
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 width,
    N_int32 height,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 rop3)
```

### Prototype In

snap/graphics.h

### Parameters

<i>srcAddr</i>	Address of source bitmap in system memory
<i>srcPhysAddr</i>	Physical address of source bitmap in system memory
<i>srcPitch</i>	Pitch of source bitmap in bytes
<i>srcLeft</i>	Left coordinate within source bitmap to copy
<i>srcTop</i>	Top coordinate within source bitmap to copy
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>rop3</i>	Microsoft ROP3 code for the copy ( <i>GA_rop3CodesType</i> )

### Description

This function is identical to the regular *BitBltSys* function, except that it also applies the currently active 8x8 monochrome pattern to the destination. The source data, pattern data and destination data are combined together according to the value passed in the *rop3* parameter.

### See Also

*BitBltPatt*, *BitBltPattLin*, *BitBltPattSys*, *SrcTransBlt*, *DstTransBlt*



## BitBltPattLin

Copy a linear block of video memory, with a mono pattern applied

### Declaration

```
void NAPI_GA_2DRenderFuncs::BitBltPattLin(
    N_int32 srcOfs,
    N_int32 srcPitch,
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 width,
    N_int32 height,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 rop3)
```

### Prototype In

snap/graphics.h

### Parameters

<i>srcOfs</i>	Offset of source bitmap in video memory
<i>srcPitch</i>	Pitch of source bitmap in bytes
<i>srcLeft</i>	Left coordinate within source bitmap to copy
<i>srcTop</i>	Top coordinate within source bitmap to copy
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>rop3</i>	Microsoft ROP3 code for the copy ( <i>GA_rop3CodesType</i> )

### Description

This function is identical to the regular *BitBltLin* function, except that it also applies the currently active 8x8 monochrome pattern to the destination. The source data, pattern data and destination data are combined together according to the value passed in the *rop3* parameter.

### See Also

*BitBltPatt*, *BitBltPattSys*, *BitBltPattBM*, *SrcTransBlt*, *DstTransBlt*

## BitBltPattSys

Copy a block of system memory to video memory, with a mono pattern applied

### Declaration

```
void NAPI_GA_2DRenderFuncs::BitBltPattSys(
    void *srcAddr,
    N_int32 srcPitch,
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 width,
    N_int32 height,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 rop3,
    N_int32 flipY)
```

### Prototype In

snap/graphics.h

### Parameters

<i>srcAddr</i>	Address of source bitmap in system memory
<i>srcPitch</i>	Pitch of source bitmap in bytes
<i>srcLeft</i>	Left coordinate within source bitmap to copy
<i>srcTop</i>	Top coordinate within source bitmap to copy
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>rop3</i>	Microsoft ROP3 code for the copy ( <i>GA_rop3CodesType</i> )
<i>flipY</i>	True if the image should be flipped vertically

### Description

This function is identical to the regular *BitBltSys* function, except that it also applies the currently active 8x8 monochrome pattern to the destination. The source data, pattern data and destination data are combined together according to the value passed in the *rop3* parameter.

### See Also

*BitBltPatt*, *BitBltPattSys*, *BitBltPattBM*, *SrcTransBlt*, *DstTransBlt*

## BitBltPlaneMasked

Copy a block of video memory with an associated bit plane mask

### Declaration

```
void NAPI_GA_2DRenderFuncs::BitBltPlaneMasked(
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 width,
    N_int32 height,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_uint32 planeMask)
```

### Prototype In

snap/graphics.h

### Parameters

<i>srcLeft</i>	Left coordinate of the source rectangle to copy
<i>srcTop</i>	Top coordinate of the source rectangle to copy
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>planeMask</i>	Plane mask to use during the copy

### Description

This function copies a rectangular region of video memory from one location to another. This routine will copy a rectangular region of video memory from (srcLeft, srcTop, srcLeft+width-1, srcTop+height-1) to (dstLeft, dstTop) within video memory with the specified plane mask. The plane mask is used to determine which bits in the destination pixels will be affected by the copy. Each bit in the plane mask is used to mask out a bit in the destination pixel values, and where a bit is a 1 the destination bit comes from the source pixel while where a bit is 0 the destination bit is left unchanged.

### See Also

*BitBlt*, *BitBltPlaneMaskedLin*, *BitBltPlaneMaskedSys*, *BitBltPlaneMaskedBM*

## BitBltPlaneMaskedBM

Copy a block of system memory to a location in video memory with Bus Mastering using an associated bit plane mask

### Declaration

```
void NAPI_GA_2DRenderFuncs::BitBltPlaneMaskedBM(
    void *srcAddr,
    N_int32 srcPhysAddr,
    N_int32 srcPitch,
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 width,
    N_int32 height,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_uint32 planeMask)
```

### Prototype In

snap/graphics.h

### Parameters

<i>srcAddr</i>	Address of source bitmap in system memory
<i>srcPhysAddr</i>	Physical address of source bitmap in system memory
<i>srcPitch</i>	Pitch of source bitmap in bytes
<i>srcLeft</i>	Left coordinate within source bitmap to copy
<i>srcTop</i>	Top coordinate within source bitmap to copy
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>planeMask</i>	Plane mask to use during the copy

### Description

This routine will copy a bitmap from system memory with a physical starting address of *srcPhysAddr* to the destination rectangle (*dstLeft*, *dstTop*, *dstLeft*+*width*-1, *dstTop*+*height*-1) with the specified plane mask. The plane mask is used to determine which bits in the destination pixels will be affected by the copy. Each bit in the plane mask is used to mask out a bit in the destination pixel values, and where a bit is a 1 the destination bit comes from the source pixel while where a bit is 0 the destination bit is left unchanged.

This version is different to the *BitBltSys* function in that the bitmap data is copied using Bus Mastering by the graphics accelerator, which allows this function to return before the copy has completed and the accelerator will complete the copy in the background with a DMA Bus Master operation. If this hardware supports Bus Mastering and this function is available, it will usually be the fastest method to copy a block of system memory to video memory.

Note that the `srcLeft` and `srcTop` coordinates define an offset within the source bitmap to be copied, so it will copy only a portion of the memory bitmap.

**See Also**

*BitBlt, BitBltPlaneMasked, BitBltPlaneMaskedLin, BitBltPlaneMaskedSys*

## BitBltPlaneMaskedLin

Copy a linear block of video memory with an associated bit plane mask

### Declaration

```
void NAPI_GA_2DRenderFuncs::BitBltPlaneMaskedLin(
    N_int32 srcOfs,
    N_int32 srcPitch,
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 width,
    N_int32 height,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_uint32 planeMask)
```

### Prototype In

snap/graphics.h

### Parameters

<i>srcOfs</i>	Offset of source bitmap in video memory
<i>srcPitch</i>	Pitch of source bitmap in bytes
<i>srcLeft</i>	Left coordinate within source bitmap to copy
<i>srcTop</i>	Top coordinate within source bitmap to copy
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>planeMask</i>	Plane mask to use during the copy

### Description

This routine will copy a linear region of video memory from *srcOfs* from the start of video memory to the destination rectangle (*dstLeft*, *dstTop*, *dstLeft*+*width*-1, *dstTop*+*height*-1) with the specified plane mask. The plane mask is used to determine which bits in the destination pixels will be affected by the copy. Each bit in the plane mask is used to mask out a bit in the destination pixel values, and where a bit is a 1 the destination bit comes from the source pixel while where a bit is 0 the destination bit is left unchanged.

Note that the *srcLeft* and *srcTop* coordinates define an offset within the source bitmap to be copied, so it is possible to only copy a portion of an offscreen bitmap. This is useful for storing multiple images in a single offscreen bitmap, or for handling the case of software clipping offscreen bitmaps if the destination lies outside of the software clip rectangle for the destination buffer.

This version is different to the standard *BitBlt* function in that the source bitmap to be copied can be non-conforming, and can have a different logical scanline width to the destination bitmap. This allows the bitmaps to be stored contiguously in offscreen video memory, rather than requiring the offscreen video memory to be divided up into

rectangular regions, resulting in more efficient use of available offscreen memory for bitmap storage.

---

**Note:** *The value of `srcOfs` must be aligned to the boundary specified in the `BitmapStartAlign` member of the `GA_modeInfo` structure, and the `srcPitch` value must be padded to multiples of the `BitmapStridePad` member of the `GA_modeInfo` structure.*

---

**See Also**

*BitBlt, BitBltPlaneMasked, BitBltPlaneMaskedSys, BitBltPlaneMaskedBM*

## BitBltPlaneMaskedSys

Copy a block of system memory to a location in video memory with an associated bit plane mask

### Declaration

```
void NAPI_GA_2DRenderFuncs::BitBltPlaneMaskedSys (
    void *srcAddr,
    N_int32 srcPitch,
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 width,
    N_int32 height,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_uint32 planeMask,
    N_int32 flipY)
```

### Prototype In

snap/graphics.h

### Parameters

<i>srcAddr</i>	Address of source bitmap in system memory
<i>srcPitch</i>	Pitch of source bitmap in bytes
<i>srcLeft</i>	Left coordinate within source bitmap to copy
<i>srcTop</i>	Top coordinate within source bitmap to copy
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>planeMask</i>	Plane mask to use during the copy
<i>flipY</i>	True if the image should be flipped vertically

### Description

This routine will copy a bitmap from system memory with a starting address of *srcAddr* to the destination rectangle (*dstLeft*, *dstTop*, *dstLeft*+*width*-1, *dstTop*+*height*-1) with the specified plane mask. The plane mask is used to determine which bits in the destination pixels will be affected by the copy. Each bit in the plane mask is used to mask out a bit in the destination pixel values, and where a bit is a 1 the destination bit comes from the source pixel while where a bit is 0 the destination bit is left unchanged.

Note that the *srcLeft* and *srcTop* coordinates define an offset within the source bitmap to be copied, so it will copy only a portion of the memory bitmap.

---

**Note:** *This routine is provided for completeness, and for the simple case of performing a system memory to video memory copy with a mix of GA\_REPLACE\_MIX, it is usually always as fast or faster to copy the bitmap data directly using a CPU memory copy directly over the system bus to the linear framebuffer. However if either hardware clipping is in use, or the mix mode is something other than GA\_REPLACE\_MIX, this function can be more efficient than doing a software only bitmap copy.*

---



**See Also**

*BitBlt, BitBltPlaneMasked, BitBltPlaneMaskedLin, BitBltPlaneMaskedBM*

## BitBltSys

Copy a block of system memory to a location in video memory.

### Declaration

```
void NAPI_GA_2DRenderFuncs::BitBltSys(
    void *srcAddr,
    N_int32 srcPitch,
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 width,
    N_int32 height,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 mix,
    N_int32 flipY)
```

### Prototype In

snap/graphics.h

### Parameters

<i>srcAddr</i>	Address of source bitmap in system memory
<i>srcPitch</i>	Pitch of source bitmap in bytes
<i>srcLeft</i>	Left coordinate within source bitmap to copy
<i>srcTop</i>	Top coordinate within source bitmap to copy
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>mix</i>	Mix code for the copy ( <i>GA_mixCodesType</i> )
<i>flipY</i>	True if the image should be flipped vertically

### Description

This routine will copy a bitmap from system memory with a starting address of *srcAddr* to the destination rectangle (*dstLeft*, *dstTop*, *dstLeft*+*width*-1, *dstTop*+*height*-1) with the specified mix. The mix code will be used to combine the source bitmap data with the pixels in the destination bitmap.

Note that the *srcLeft* and *srcTop* coordinates define an offset within the source bitmap to be copied, so it will copy only a portion of the memory bitmap.

---

**Note:** This routine is provided for completeness, and for the simple case of performing a system memory to video memory copy with a mix of *GA\_REPLACE\_MIX*, it is usually always as fast or faster to copy the bitmap data directly using a CPU memory copy directly over the system bus to the linear framebuffer. However if either hardware clipping is in use, or the mix mode is something other than *GA\_REPLACE\_MIX*, this function can be more efficient than doing a software only bitmap copy.

---

### See Also

*BitBlt*, *BitBltSys*, *BitBltBM*, *SrcTransBlt*, *DstTransBlt*, *BitBltFx*

## ClipEllipse

Draw a clipped, single pixel wide, outlined ellipse

### Declaration

```
void NAPI_GA_2DRenderFuncs::ClipEllipse(
    N_int32 left,
    N_int32 top,
    N_int32 A,
    N_int32 B,
    N_int32 clipLeft,
    N_int32 clipTop,
    N_int32 clipRight,
    N_int32 clipBottom)
```

### Prototype In

snap/graphics.h

### Parameters

<i>left</i>	Left coordinate for first pixel in the ellipse
<i>top</i>	Top coordinate for first pixel in the ellipse
<i>A</i>	Major axis dimension
<i>B</i>	Minor axis dimension
<i>clipLeft</i>	Left coordinate for clip rectangle (inclusive)
<i>clipTop</i>	Top coordinate for clip rectangle (inclusive)
<i>clipRight</i>	Right coordinate for clip rectangle (exclusive)
<i>clipBottom</i>	Bottom coordinate for clip rectangle (exclusive)

### Description

This function draws a single pixel wide, outlined ellipse using the currently active foreground color and mix. The output is clipped against the passed in clip rectangle.

### See Also

*DrawEllipse*

## ClipMonoImageLSBBM

Draws a monochrome bitmap stored in system memory with bus mastering and clipping

### Declaration

```
void NAPI_GA_2DRenderFuncs::ClipMonoImageLSBBM(
    N_int32 x,
    N_int32 y,
    N_int32 width,
    N_int32 height,
    N_int32 byteWidth,
    N_uint8 *image,
    N_int32 imagePhysAddr,
    N_int32 transparent,
    N_int32 clipLeft,
    N_int32 clipTop,
    N_int32 clipRight,
    N_int32 clipBottom)
```

### Prototype In

snap/graphics.h

### Parameters

<i>x</i>	Destination X coordinate to draw the bitmap at
<i>y</i>	Destination Y coordinate to draw the bitmap at
<i>width</i>	Width of the bitmap in pixels
<i>height</i>	Height of the bitmap in pixels
<i>byteWidth</i>	Width of the bitmap in bytes
<i>image</i>	Pointer to the bitmap image data to draw
<i>imagePhysAddr</i>	Physical address of bitmap image data in system memory
<i>transparent</i>	1 for transparent, 0 for opaque
<i>clipLeft</i>	Left coordinate for clip rectangle
<i>clipTop</i>	Top coordinate for clip rectangle
<i>clipRight</i>	Right coordinate for clip rectangle (exclusive)
<i>clipBottom</i>	Bottom coordinate for clip rectangle (exclusive)

### Description

This function is identical to the equivalent PutMonoImage\* family function, except that it takes a clip rectangle and will clip the output of the monochrome image to the specified clip rectangle.

### See Also

*ClipMonoImageLSBSys, ClipMonoImageLSBBM, ClipMonoImageMSBSys, ClipMonoImageMSBLin, ClipMonoImageMSBBM, PutMonoImageLSBSys, PutMonoImageLSBLin, PutMonoImageLSBBM, PutMonoImageMSBSys, PutMonoImageMSBLin, PutMonoImageMSBBM*

## ClipMonoImageLSBLin

Draws a monochrome bitmap stored in offscreen video memory, with clipping.

### Declaration

```
void NAPI_GA_2DRenderFuncs::ClipMonoImageLSBLin(
    N_int32 x,
    N_int32 y,
    N_int32 width,
    N_int32 height,
    N_int32 byteWidth,
    N_int32 imageOfs,
    N_int32 transparent,
    N_int32 clipLeft,
    N_int32 clipTop,
    N_int32 clipRight,
    N_int32 clipBottom)
```

### Prototype In

snap/graphics.h

### Parameters

<i>x</i>	Destination X coordinate to draw the bitmap at
<i>y</i>	Destination Y coordinate to draw the bitmap at
<i>width</i>	Width of the bitmap in pixels
<i>height</i>	Height of the bitmap in pixels
<i>byteWidth</i>	Width of the bitmap in bytes
<i>imageOfs</i>	Offset of bitmap image data in video memory (byte address)
<i>transparent</i>	1 for transparent, 0 for opaque
<i>clipLeft</i>	Left coordinate for clip rectangle
<i>clipTop</i>	Top coordinate for clip rectangle
<i>clipRight</i>	Right coordinate for clip rectangle (exclusive)
<i>clipBottom</i>	Bottom coordinate for clip rectangle (exclusive)

### Description

This function is identical to the equivalent PutMonoImage\* family function, except that it takes a clip rectangle and will clip the output of the monochrome image to the specified clip rectangle.

### See Also

*ClipMonoImageLSBSys, ClipMonoImageLSBBM, ClipMonoImageMSBSys, ClipMonoImageMSBLin, ClipMonoImageMSBBM, PutMonoImageLSBSys, PutMonoImageLSBLin, PutMonoImageLSBBM, PutMonoImageMSBSys, PutMonoImageMSBLin, PutMonoImageMSBBM*

## ClipMonoImageLSBSys

Draws a monochrome bitmap stored in system memory, with clipping.

### Declaration

```
void NAPI_GA_2DRenderFuncs::ClipMonoImageLSBSys (
    N_int32 x,
    N_int32 y,
    N_int32 width,
    N_int32 height,
    N_int32 byteWidth,
    N_uint8 *image,
    N_int32 transparent,
    N_int32 clipLeft,
    N_int32 clipTop,
    N_int32 clipRight,
    N_int32 clipBottom)
```

### Prototype In

snap/graphics.h

### Parameters

<i>x</i>	Destination X coordinate to draw the bitmap at
<i>y</i>	Destination Y coordinate to draw the bitmap at
<i>width</i>	Width of the bitmap in pixels
<i>height</i>	Height of the bitmap in pixels
<i>byteWidth</i>	Width of the bitmap in bytes
<i>image</i>	Pointer to the bitmap image data to draw
<i>transparent</i>	1 for transparent, 0 for opaque
<i>clipLeft</i>	Left coordinate for clip rectangle
<i>clipTop</i>	Top coordinate for clip rectangle
<i>clipRight</i>	Right coordinate for clip rectangle (exclusive)
<i>clipBottom</i>	Bottom coordinate for clip rectangle (exclusive)

### Description

This function is identical to the equivalent PutMonoImage\* family function, except that it takes a clip rectangle and will clip the output of the monochrome image to the specified clip rectangle.

### See Also

*ClipMonoImageLSBLin*, *ClipMonoImageLSBBM*, *ClipMonoImageMSBSys*,  
*ClipMonoImageMSBLin*, *ClipMonoImageMSBBM*, *PutMonoImageLSBSys*,  
*PutMonoImageLSBLin*, *PutMonoImageLSBBM*, *PutMonoImageMSBSys*,  
*PutMonoImageMSBLin*, *PutMonoImageMSBBM*

## ClipMonoImageMSBBM

Draws a monochrome bitmap stored in system memory with bus mastering and clipping.

### Declaration

```
void NAPI_GA_2DRenderFuncs::ClipMonoImageMSBBM(
    N_int32 x,
    N_int32 y,
    N_int32 width,
    N_int32 height,
    N_int32 byteWidth,
    N_uint8 *image,
    N_int32 imagePhysAddr,
    N_int32 transparent,
    N_int32 clipLeft,
    N_int32 clipTop,
    N_int32 clipRight,
    N_int32 clipBottom)
```

### Prototype In

snap/graphics.h

### Parameters

<i>x</i>	Destination X coordinate to draw the bitmap at
<i>y</i>	Destination Y coordinate to draw the bitmap at
<i>width</i>	Width of the bitmap in pixels
<i>height</i>	Height of the bitmap in pixels
<i>byteWidth</i>	Width of the bitmap in bytes
<i>image</i>	Pointer to the bitmap image data to draw
<i>imagePhysAddr</i>	Physical address of bitmap image data in system memory
<i>transparent</i>	1 for transparent, 0 for opaque
<i>clipLeft</i>	Left coordinate for clip rectangle
<i>clipTop</i>	Top coordinate for clip rectangle
<i>clipRight</i>	Right coordinate for clip rectangle (exclusive)
<i>clipBottom</i>	Bottom coordinate for clip rectangle (exclusive)

### Description

This function is identical to the equivalent PutMonoImage\* family function, except that it takes a clip rectangle and will clip the output of the monochrome image to the specified clip rectangle.

### See Also

*ClipMonoImageMSBSys, ClipMonoImageMSBBM, ClipMonoImageMSBSys, ClipMonoImageLSBLin, ClipMonoImageLSBBM, PutMonoImageLSBSys, PutMonoImageLSBLin, PutMonoImageLSBBM, PutMonoImageMSBSys, PutMonoImageMSBLin, PutMonoImageMSBBM*

## ClipMonoImageMSBLin

Draws a monochrome bitmap stored in offscreen video memory, with clipping.

### Declaration

```
void NAPI_GA_2DRenderFuncs::ClipMonoImageMSBLin(
    N_int32 x,
    N_int32 y,
    N_int32 width,
    N_int32 height,
    N_int32 byteWidth,
    N_int32 imageOfs,
    N_int32 transparent,
    N_int32 clipLeft,
    N_int32 clipTop,
    N_int32 clipRight,
    N_int32 clipBottom)
```

### Prototype In

snap/graphics.h

### Parameters

<i>x</i>	Destination X coordinate to draw the bitmap at
<i>y</i>	Destination Y coordinate to draw the bitmap at
<i>width</i>	Width of the bitmap in pixels
<i>height</i>	Height of the bitmap in pixels
<i>byteWidth</i>	Width of the bitmap in bytes
<i>imageOfs</i>	Offset of bitmap image data in video memory (byte address)
<i>transparent</i>	1 for transparent, 0 for opaque
<i>clipLeft</i>	Left coordinate for clip rectangle
<i>clipTop</i>	Top coordinate for clip rectangle
<i>clipRight</i>	Right coordinate for clip rectangle (exclusive)
<i>clipBottom</i>	Bottom coordinate for clip rectangle (exclusive)

### Description

This function is identical to the equivalent PutMonoImage\* family function, except that it takes a clip rectangle and will clip the output of the monochrome image to the specified clip rectangle.

### See Also

*ClipMonoImageMSBSys, ClipMonoImageMSBBM, ClipMonoImageMSBSys, ClipMonoImageLSBLin, ClipMonoImageLSBBM, PutMonoImageLSBSys, PutMonoImageLSBLin, PutMonoImageLSBBM, PutMonoImageMSBSys, PutMonoImageMSBLin, PutMonoImageMSBBM*



## ClipMonoImageMSBSys

Draws a monochrome bitmap stored in system memory, with clipping.

### Declaration

```
void NAPI_GA_2DRenderFuncs::ClipMonoImageMSBSys (
    N_int32 x,
    N_int32 y,
    N_int32 width,
    N_int32 height,
    N_int32 byteWidth,
    N_uint8 *image,
    N_int32 transparent,
    N_int32 clipLeft,
    N_int32 clipTop,
    N_int32 clipRight,
    N_int32 clipBottom)
```

### Prototype In

snap/graphics.h

### Parameters

<i>x</i>	Destination X coordinate to draw the bitmap at
<i>y</i>	Destination Y coordinate to draw the bitmap at
<i>width</i>	Width of the bitmap in pixels
<i>height</i>	Height of the bitmap in pixels
<i>byteWidth</i>	Width of the bitmap in bytes
<i>image</i>	Pointer to the bitmap image data to draw
<i>transparent</i>	1 for transparent, 0 for opaque
<i>clipLeft</i>	Left coordinate for clip rectangle
<i>clipTop</i>	Top coordinate for clip rectangle
<i>clipRight</i>	Right coordinate for clip rectangle (exclusive)
<i>clipBottom</i>	Bottom coordinate for clip rectangle (exclusive)

### Description

This function is identical to the equivalent PutMonoImage\* family function, except that it takes a clip rectangle and will clip the output of the monochrome image to the specified clip rectangle.

### See Also

*ClipMonoImageMSBLin*, *ClipMonoImageMSBBM*, *ClipMonoImageMSBSys*,  
*ClipMonoImageLSBLin*, *ClipMonoImageLSBBM*, *SetMonoClipRect*, *PutMonoImageLSBSys*,  
*PutMonoImageLSBLin*, *PutMonoImageLSBBM*, *PutMonoImageMSBSys*,  
*PutMonoImageMSBLin*, *PutMonoImageMSBBM*

## DrawBresenhamLine

Draws a solid, single pixel wide line with bresenham parameters.

### Declaration

```
void NAPI_GA_2DRenderFuncs::DrawBresenhamLine(
    N_int32 x1,
    N_int32 y1,
    N_int32 initialError,
    N_int32 majorInc,
    N_int32 diagInc,
    N_int32 count,
    N_int32 flags)
```

### Prototype In

snap/graphics.h

### Parameters

<i>x1</i>	X1 coordinate
<i>y1</i>	Y1 coordinate
<i>initialError</i>	Initial error term
<i>majorInc</i>	Major increment factor for error term
<i>diagInc</i>	Diagonal increment factor for error term
<i>count</i>	Number of pixels to draw
<i>flags</i>	Flags to control drawing ( <i>GA_BresenhamLineFlagsType</i> )

### Description

This function is similar to the regular *DrawLineInt* function, except that the bresenham parameters for the line are pre-computed and passed to this function. Allowing the application to pre-compute the bresenham parameters allows for accurate clipping as the bresenham parameters can be computed for the unclipped line, and initialised to the start of the clipped line segment. This function is used by OS/2 display drivers and the SciTech MGL to get accurate line clipping. The values passed to the function for a regular integer line can be computed as follows:

```
flags = gaLineXPositive | gaLineYPositive | gaLineXMajor
      | gaLineDoLastPel;
if ((absDeltaX = x2 - x1) < 0) {
    absDeltaX = -absDeltaX;
    flags &= ~gaLineXPositive;
}
if ((absDeltaY = y2 - y1) < 0) {
    absDeltaY = -absDeltaY;
    flags &= ~gaLineYPositive;
}
if (absDeltaY > absDeltaX) {
    SWAP(absDeltaX, absDeltaY);
    flags &= ~gaLineXMajor;
}
majorInc = 2 * absDeltaY;           // 2 * dy
initialError = majorInc - absDeltaX; // 2 * dy - dx
diagInc = initialError - absDeltaX; // 2 * (dy - dx)
```

### See Also

*DrawLineInt, DrawStippleLineInt*

## DrawBresenhamStippleLine

Draws a stippled, single pixel wide line with bresenham parameters.

### Declaration

```
void NAPI_GA_2DRenderFuncs::DrawBresenhamStippleLine(
    N_int32 x1,
    N_int32 y1,
    N_int32 initialError,
    N_int32 majorInc,
    N_int32 diagInc,
    N_int32 count,
    N_int32 flags,
    N_int32 transparent)
```

### Prototype In

snap/graphics.h

### Parameters

<i>x1</i>	X1 coordinate
<i>y1</i>	Y1 coordinate
<i>initialError</i>	Initial error term
<i>majorInc</i>	Major increment factor for error term
<i>diagInc</i>	Diagonal increment factor for error term
<i>count</i>	Number of pixels to draw
<i>flags</i>	Flags to control drawing ( <i>GA_BresenhamLineFlagsType</i> )
<i>transparent</i>	True for a transparent background

### Description

This function is similar to the regular *DrawStippleLineInt* function, except that the bresenham parameters for the line are pre-computed and passed to this function. Allowing the application to pre-compute the bresenham parameters allows for accurate clipping as the bresenham parameters can be computed for the unclipped line, and initialised to the start of the clipped line segment. This function is used by OS/2 display drivers to get accurate line clipping. The values passed to the function for a regular integer line can be computed as follows:

```
flags = gaLineXPositive | gaLineYPositive | gaLineXMajor
      | gaLineDoLastPel;
if ((absDeltaX = x2 - x1) < 0) {
    absDeltaX = -absDeltaX;
    flags &= ~gaLineXPositive;
}
if ((absDeltaY = y2 - y1) < 0) {
    absDeltaY = -absDeltaY;
    flags &= ~gaLineYPositive;
}
if (absDeltaY > absDeltaX) {
    SWAP(absDeltaX, absDeltaY);
    flags &= ~gaLineXMajor;
}
majorInc = 2 * absDeltaY;           // 2 * dy
initialError = majorInc - absDeltaX; // 2 * dy - dx
diagInc = initialError - absDeltaX; // 2 * (dy - dx)
```

**See Also**

*DrawStippleLineInt, DrawLineInt, SetLineStipple, SetLineStippleCount*

## DrawBresenhamStyleLine

Draws an OS/2 style styled, single pixel wide line with bresenham parameters.

### Declaration

```
void NAPI_GA_2DRenderFuncs::DrawBresenhamStyleLine(
    N_int32 x1,
    N_int32 y1,
    N_int32 initialError,
    N_int32 majorInc,
    N_int32 diagInc,
    N_int32 count,
    N_int32 flags,
    N_int32 transparent)
```

### Prototype In

snap/graphics.h

### Parameters

<i>x1</i>	X1 coordinate
<i>y1</i>	Y1 coordinate
<i>initialError</i>	Initial error term
<i>majorInc</i>	Major increment factor for error term
<i>diagInc</i>	Diagonal increment factor for error term
<i>count</i>	Number of pixels to draw
<i>flags</i>	Flags to control drawing ( <i>GA_BresenhamLineFlagsType</i> )
<i>transparent</i>	True for a transparent background

### Description

This function is similar to the regular *DrawStippleLineInt* function, except that the bresenham parameters for the line are pre-computed and passed to this function. Allowing the application to pre-compute the bresenham parameters allows for accurate clipping as the bresenham parameters can be computed for the unclipped line, and initialised to the start of the clipped line segment. This function is used by OS/2 display drivers to get accurate line clipping. The values passed to the function for a regular integer line can be computed as follows:

```
flags = gaLineXPositive | gaLineYPositive | gaLineXMajor
      | gaLineDoLastPel;
if ((absDeltaX = x2 - x1) < 0) {
    absDeltaX = -absDeltaX;
    flags &= ~gaLineXPositive;
}
if ((absDeltaY = y2 - y1) < 0) {
    absDeltaY = -absDeltaY;
    flags &= ~gaLineYPositive;
}
if (absDeltaY > absDeltaX) {
    SWAP(absDeltaX, absDeltaY);
    flags &= ~gaLineXMajor;
}
majorInc = 2 * absDeltaY;           // 2 * dy
initialError = majorInc - absDeltaX; // 2 * dy - dx
diagInc = initialError - absDeltaX; // 2 * (dy - dx)
```

**See Also**

*DrawStyleLineInt, DrawLineInt, SetLineStyle*

## DrawClippedBresenhamLine

Draws a solid, single pixel wide line with bresenham parameters.

### Declaration

```
N_int32 NAPI GA_2DRenderFuncs::DrawClippedBresenhamLine (
    N_int32 x1,
    N_int32 y1,
    N_int32 initialError,
    N_int32 majorInc,
    N_int32 diagInc,
    N_int32 count,
    N_int32 flags,
    N_int32 clipLeft,
    N_int32 clipTop,
    N_int32 clipRight,
    N_int32 clipBottom)
```

### Prototype In

snap/graphics.h

### Parameters

<i>x1</i>	X1 coordinate
<i>y1</i>	Y1 coordinate
<i>initialError</i>	Initial error term
<i>majorInc</i>	Major increment factor for error term
<i>diagInc</i>	Diagonal increment factor for error term
<i>count</i>	Number of pixels to draw
<i>flags</i>	Flags to control drawing ( <i>GA_BresenhamLineFlagsType</i> )
<i>clipLeft</i>	Left coordinate for clip rectangle (inclusive)
<i>clipTop</i>	Top coordinate for clip rectangle (inclusive)
<i>clipRight</i>	Right coordinate for clip rectangle (exclusive)
<i>clipBottom</i>	Bottom coordinate for clip rectangle (exclusive)

### Description

This function is similar to the regular *DrawLineInt* function, except that the bresenham parameters for the line are pre-computed and passed to this function. Allowing the application to pre-compute the bresenham parameters allows for accurate clipping as the bresenham parameters can be computed for the unclipped line, and initialised to the start of the clipped line segment. This function is used by OS/2 display drivers and the SciTech MGL to get accurate line clipping. The values passed to the function for a regular integer line can be computed as follows:

```
flags = gaLineXPositive | gaLineYPositive | gaLineXMajor
      | gaLineDoLastPel;
if ((absDeltaX = x2 - x1) < 0) {
    absDeltaX = -absDeltaX;
    flags &= ~gaLineXPositive;
}
if ((absDeltaY = y2 - y1) < 0) {
    absDeltaY = -absDeltaY;
    flags &= ~gaLineYPositive;
}
```



```

if (absDeltaY > absDeltaX) {
    SWAP(absDeltaX,absDeltaY);
    flags &= ~gaLineXMajor;
}
majorInc = 2 * absDeltaY;           // 2 * dy
initialError = majorInc - absDeltaX; // 2 * dy - dx
diagInc = initialError - absDeltaX; // 2 * (dy - dx)

```

The output is clipped against the passed in clipping rectangle.

### See Also

*DrawClippedLineInt, DrawClippedStippleLineInt*

## DrawClippedBresenhamStippleLine

Draws a stippled, single pixel wide line with bresenham parameters.

### Declaration

```
N_int32 NAPI GA_2DRenderFuncs::DrawClippedBresenhamStippleLine(
    N_int32 x1,
    N_int32 y1,
    N_int32 initialError,
    N_int32 majorInc,
    N_int32 diagInc,
    N_int32 count,
    N_int32 flags,
    N_int32 transparent,
    N_int32 clipLeft,
    N_int32 clipTop,
    N_int32 clipRight,
    N_int32 clipBottom)
```

### Prototype In

snap/graphics.h

### Parameters

<i>x1</i>	X1 coordinate
<i>y1</i>	Y1 coordinate
<i>initialError</i>	Initial error term
<i>majorInc</i>	Major increment factor for error term
<i>diagInc</i>	Diagonal increment factor for error term
<i>count</i>	Number of pixels to draw
<i>flags</i>	Flags to control drawing ( <i>GA_BresenhamLineFlagsType</i> )
<i>transparent</i>	True for a transparent background
<i>clipLeft</i>	Left coordinate for clip rectangle (inclusive)
<i>clipTop</i>	Top coordinate for clip rectangle (inclusive)
<i>clipRight</i>	Right coordinate for clip rectangle (exclusive)
<i>clipBottom</i>	Bottom coordinate for clip rectangle (exclusive)

### Description

This function is similar to the regular *DrawStippleLineInt* function, except that the bresenham parameters for the line are pre-computed and passed to this function. Allowing the application to pre-compute the bresenham parameters allows for accurate clipping as the bresenham parameters can be computed for the unclipped line, and initialised to the start of the clipped line segment. This function is used by OS/2 display drivers to get accurate line clipping. The values passed to the function for a regular integer line can be computed as follows:

```
flags = gaLineXPositive | gaLineYPositive | gaLineXMajor
      | gaLineDoLastPel;
if ((absDeltaX = x2 - x1) < 0) {
    absDeltaX = -absDeltaX;
    flags &= ~gaLineXPositive;
}
if ((absDeltaY = y2 - y1) < 0) {
```

```

        absDeltaY = -absDeltaY;
        flags &= ~gaLineYPositive;
    }
    if (absDeltaY > absDeltaX) {
        SWAP(absDeltaX,absDeltaY);
        flags &= ~gaLineXMajor;
    }
    majorInc = 2 * absDeltaY;           // 2 * dy
    initialError = majorInc - absDeltaX; // 2 * dy - dx
    diagInc = initialError - absDeltaX; // 2 * (dy - dx)

```

The output is clipped against the passed in clipping rectangle.

### See Also

*DrawClippedStippleLineInt, DrawClippedLineInt, SetLineStipple, SetLineStippleCount*

## DrawClippedBresenhamStyleLine

Draws an OS/2 style styled, single pixel wide line with bresenham parameters.

### Declaration

```
N_int32 NAPI GA_2DRenderFuncs::DrawClippedBresenhamStyleLine(
    N_int32 x1,
    N_int32 y1,
    N_int32 initialError,
    N_int32 majorInc,
    N_int32 diagInc,
    N_int32 count,
    N_int32 flags,
    N_int32 transparent,
    N_int32 clipLeft,
    N_int32 clipTop,
    N_int32 clipRight,
    N_int32 clipBottom)
```

### Prototype In

snap/graphics.h

### Parameters

<i>x1</i>	X1 coordinate
<i>y1</i>	Y1 coordinate
<i>initialError</i>	Initial error term
<i>majorInc</i>	Major increment factor for error term
<i>diagInc</i>	Diagonal increment factor for error term
<i>count</i>	Number of pixels to draw
<i>flags</i>	Flags to control drawing ( <i>GA_BresenhamLineFlagsType</i> )
<i>transparent</i>	True for a transparent background
<i>clipLeft</i>	Left coordinate for clip rectangle (inclusive)
<i>clipTop</i>	Top coordinate for clip rectangle (inclusive)
<i>clipRight</i>	Right coordinate for clip rectangle (exclusive)
<i>clipBottom</i>	Bottom coordinate for clip rectangle (exclusive)

### Description

This function is similar to the regular *DrawStippleLineInt* function, except that the bresenham parameters for the line are pre-computed and passed to this function. Allowing the application to pre-compute the bresenham parameters allows for accurate clipping as the bresenham parameters can be computed for the unclipped line, and initialised to the start of the clipped line segment. This function is used by OS/2 display drivers to get accurate line clipping. The values passed to the function for a regular integer line can be computed as follows:

```
flags = gaLineXPositive | gaLineYPositive | gaLineXMajor
      | gaLineDoLastPel;
if ((absDeltaX = x2 - x1) < 0) {
    absDeltaX = -absDeltaX;
    flags &= ~gaLineXPositive;
}
if ((absDeltaY = y2 - y1) < 0) {
```

```

        absDeltaY = -absDeltaY;
        flags &= ~gaLineYPositive;
    }
    if (absDeltaY > absDeltaX) {
        SWAP(absDeltaX,absDeltaY);
        flags &= ~gaLineXMajor;
    }
    majorInc = 2 * absDeltaY;           // 2 * dy
    initialError = majorInc - absDeltaX; // 2 * dy - dx
    diagInc = initialError - absDeltaX; // 2 * (dy - dx)

```

The output is clipped against the passed in clipping rectangle.

### See Also

*DrawClippedStyleLineInt, DrawClippedLineInt, SetLineStyle*

## DrawClippedLineInt

Draws a solid, single pixel wide line with integer coordinates.

### Declaration

```
N_int32 NAPI GA_2DRenderFuncs::DrawClippedLineInt(
    N_int32 x1,
    N_int32 y1,
    N_int32 x2,
    N_int32 y2,
    N_int32 drawLast,
    N_int32 clipLeft,
    N_int32 clipTop,
    N_int32 clipRight,
    N_int32 clipBottom)
```

### Prototype In

snap/graphics.h

### Parameters

<i>x1</i>	X1 coordinate
<i>y1</i>	Y1 coordinate
<i>x2</i>	X2 coordinate
<i>y2</i>	Y2 coordinate
<i>drawLast</i>	1 to draw last pixel, 0 to skip it
<i>clipLeft</i>	Left coordinate for clip rectangle (inclusive)
<i>clipTop</i>	Top coordinate for clip rectangle (inclusive)
<i>clipRight</i>	Right coordinate for clip rectangle (exclusive)
<i>clipBottom</i>	Bottom coordinate for clip rectangle (exclusive)

### Description

This function renders a solid line at the specified location and the currently active color and mix, clipping to the pass in clip rectangle. This function is really intended as a fast way to handle clipped lines if the hardware can do hardware clipping, and to allow filters drivers to handle clipping efficiently (ie: multi controller, portrait etc).

The output is clipped against the passed in clipping rectangle.

### See Also

*DrawClippedBresenhamLine*, *DrawClippedStippleLine*, *DrawClippedStippleLineInt*, *DrawDrawClippedStyleLineInt*

## DrawClippedStippleLineInt

Draws a stippled, single pixel wide line with integer coordinates.

### Declaration

```
N_int32 NAPI GA_2DRenderFuncs::DrawClippedStippleLineInt (
    N_int32 x1,
    N_int32 y1,
    N_int32 x2,
    N_int32 y2,
    N_int32 drawLast,
    N_int32 transparent,
    N_int32 clipLeft,
    N_int32 clipTop,
    N_int32 clipRight,
    N_int32 clipBottom)
```

### Prototype In

snap/graphics.h

### Parameters

<i>x1</i>	X1 coordinate
<i>y1</i>	Y1 coordinate
<i>x2</i>	X2 coordinate
<i>y2</i>	Y2 coordinate
<i>drawLast</i>	1 to draw last pixel, 0 to skip it
<i>transparent</i>	1 if the line is transparent, 0 if opaque
<i>clipLeft</i>	Left coordinate for clip rectangle (inclusive)
<i>clipTop</i>	Top coordinate for clip rectangle (inclusive)
<i>clipRight</i>	Right coordinate for clip rectangle (exclusive)
<i>clipBottom</i>	Bottom coordinate for clip rectangle (exclusive)

### Description

This function renders a stippled line at the specified location and the currently active colors, mix and stipple pattern. This routine will render a line from (x1,y1) to (x2,y2) inclusive. If the drawLast parameter is set, the last pixel in the line (x2,y2) will be drawn, otherwise it will be skipped. This feature allows multiple lines to be linked together as a polyline for CAD style operations while drawing in XOR mode (and is also required for compatibility with Microsoft Windows).

If the transparent parameter is set to 1, where a bit is 0 in the stipple pattern the destination pixel remains untouched. If the transparent parameter is set to 0, where a bit is 0 in the stipple pattern the destination pixel is drawn in the background color. In all cases where a bit in the stipple pattern is 1, the pixel is drawn in the foreground color.

The output is clipped against the passed in clipping rectangle.

### See Also

*DrawClippedBresenhamStippleLine, DrawClippedLineInt, SetLineStipple, SetLineStippleCount*

## DrawClippedStyleLineInt

Draws an OS/2 style styled, single pixel wide line with integer coordinates.

### Declaration

```
N_int32 NAPI GA_2DRenderFuncs::DrawClippedStyleLineInt (
    N_int32 x1,
    N_int32 y1,
    N_int32 x2,
    N_int32 y2,
    N_int32 drawLast,
    N_int32 transparent,
    N_int32 clipLeft,
    N_int32 clipTop,
    N_int32 clipRight,
    N_int32 clipBottom)
```

### Prototype In

snap/graphics.h

### Parameters

<i>x1</i>	X1 coordinate
<i>y1</i>	Y1 coordinate
<i>x2</i>	X2 coordinate
<i>y2</i>	Y2 coordinate
<i>drawLast</i>	1 to draw last pixel, 0 to skip it
<i>transparent</i>	1 if the line is transparent, 0 if opaque
<i>clipLeft</i>	Left coordinate for clip rectangle (inclusive)
<i>clipTop</i>	Top coordinate for clip rectangle (inclusive)
<i>clipRight</i>	Right coordinate for clip rectangle (exclusive)
<i>clipBottom</i>	Bottom coordinate for clip rectangle (exclusive)

### Description

This function renders an OS/2 style styled line at the specified location and the currently active colors, mix and style pattern. This routine will render a line from (x1,y1) to (x2,y2) inclusive. If the drawLast parameter is set, the last pixel in the line (x2,y2) will be drawn, otherwise it will be skipped. This feature allows multiple lines to be linked together as a polyline for CAD style operations while drawing in XOR mode (and is also required for compatibility with Microsoft Windows).

If the transparent parameter is set to 1, where a bit is 0 in the style pattern the destination pixel remains untouched. If the transparent parameter is set to 0, where a bit is 0 in the style pattern the destination pixel is drawn in the background color. In all cases where a bit in the style pattern is 1, the pixel is drawn in the foreground color.

The output is clipped against the passed in clipping rectangle.

### See Also

*DrawClippedBresenhamStyleLine, DrawClippedLineInt, SetLineStyle*



## DrawColorPattEllipseList

Draws a list of color patterned scanlines for a filled ellipse engine back end.

### Declaration

```
void NAPI_GA_2DRenderFuncs::DrawColorPattEllipseList (
    N_int32 y,
    N_int32 length,
    N_int32 height,
    N_int16 *scans)
```

### Prototype In

snap/graphics.h

### Parameters

<i>y</i>	Y coordinate for scanline
<i>length</i>	Number of scanlines in the list
<i>height</i>	Height of the ellipse minor axis
<i>scans</i>	Pointer to an array of scanline data

### Description

This function renders a list of color patterned scanlines starting at the specified location in the currently active mix and color pattern. This function forms the back end of a fast filled ellipse rendering engine, but does not actually compute the scanlines in the list itself since the pixelisation rules are usually different for different device driver environments.

The scanline coordinates are passed as an array of 16-bit integer coordinates, packed with the LEFT coordinate followed by the RIGHT coordinate and so on for each scanline. For each scanline in the list, this routine will render a scanline from LEFT to RIGHT (exclusive) at increasing Y coordinates. The calling code must always guarantee that the LEFT coordinates will be less than the RIGHT coordinates, and that they will never be equal for each scanline.

The algorithm used internally in the drivers to render the list of scanlines is similar to the following:

```
maxIndex = length-1;
for (i = 0, j = height; i < maxIndex; i++, j--, scans += 2) {
    ColorPattScan(i, scans[0], scans[1]);
    ColorPattScan(j, scans[0], scans[1]);
}
if (!(height & 1))
    ColorPattScan(i, scans[0], scans[0]);
```

### See Also

*DrawEllipseList, DrawPattEllipseList, DrawFatEllipseList, DrawPattFatEllipseList, DrawColorPattFatEllipseList*

## DrawColorPattFatEllipseList

Draws a list of color patterned scanlines for a fat ellipse engine back end.

### Declaration

```
void NAPI_GA_2DRenderFuncs::DrawColorPattFatEllipseList(
    N_int32 y,
    N_int32 length,
    N_int32 height,
    N_int16 *scans)
```

### Prototype In

snap/graphics.h

### Parameters

<i>y</i>	Y coordinate for scanline
<i>length</i>	Number of scanlines in the list
<i>height</i>	Height of the ellipse minor axis + pen height adjustment
<i>scans</i>	Pointer to an array of scanline data

### Description

This function renders a list of color patterned scanlines starting at the specified location in the currently active mix and color pattern. This function forms the back end of a fast fat pen ellipse rendering engine, but does not actually compute the scanlines in the list itself since the pixelisation rules are usually different for different device driver environments.

The scanline coordinates are passed as an array of 16-bit integer coordinates, packed in multiples of 4 coordinates for a single scanline list. The first coordinate is the LEFTL coordinate, the second is the LEFTR, the third is the RIGHTL and the fourth is the RIGHTR coordinate. For each scanline in the list (each list defines two scanlines at the same Y coordinate), this routine will render a scanline from LEFT to RIGHT (exclusive) at increasing Y coordinates. The calling code must always guarantee that the LEFT coordinates will be less than the RIGHT coordinates, and that they will never be equal for each scanline.

The algorithm used internally in the drivers to render the list of scanlines is similar to the following:

```
for (i = 0, j = height; i < length; i++, j--, scans += 4) {
    if (scans[LEFTR] < scans[RIGHTL]) {
        ColorPattScan(i, scans[LEFTL], scans[LEFTR]);
        ColorPattScan(i, scans[RIGHTL], scans[RIGHTR]);
        ColorPattScan(j, scans[LEFTL], scans[LEFTR]);
        ColorPattScan(j, scans[RIGHTL], scans[RIGHTR]);
    }
    else {
        ColorPattScan(i, scans[LEFTL], scans[RIGHTR]);
        ColorPattScan(j, scans[LEFTL], scans[RIGHTR]);
    }
}
if ((height+1) & 1) {
```

```
if (scans[LEFTR] < scans[RIGHTL]) {  
    ColorPattScan(i, scans[LEFTL], scans[LEFTR]);  
    ColorPattScan(i, scans[RIGHTL], scans[RIGHTR]);  
}  
else {  
    ColorPattScan(i, scans[LEFTL], scans[RIGHTR]);  
}  
}
```

**See Also**

*DrawEllipseList, DrawPattEllipseList, DrawColorPattEllipseList, DrawFatEllipseList,  
DrawPattFatEllipseList*

## DrawColorPattRect

Draws a color pattern filled rectangle.

### Declaration

```
void NAPI_GA_2DRenderFuncs::DrawColorPattRect (
    N_int32 left,
    N_int32 top,
    N_int32 width,
    N_int32 height)
```

### Prototype In

snap/graphics.h

### Parameters

<i>left</i>	Left coordinate of the rectangle to draw
<i>top</i>	Top coordinate of the rectangle to draw
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines

### Description

This function renders a monochrome patterned rectangle at the specified location in the currently active mix and color pattern. This routine will render a rectangle from (Left, Top) to (Left+Width-1, Height+Bottom-1) inclusive.

### See Also

*DrawRect, DrawPattRect, Set8x8ColorPattern, Use8x8ColorPattern*

## DrawColorPattScanList

Draws a list of color patterned scanlines.

### Declaration

```
void NAPI_GA_2DRenderFuncs::DrawColorPattScanList(
    N_int32 y,
    N_int32 length,
    N_int16 *scans)
```

### Prototype In

snap/graphics.h

### Parameters

<i>y</i>	Y coordinate for scanline
<i>length</i>	Number of scanlines in the list
<i>scans</i>	Pointer to an array of scanline data

### Description

This function renders a list of color patterned scanlines starting at the specified location with the currently active mix and color pattern. The scanline coordinates are passed as an array of 16-bit integer coordinates, packed with the X1 coordinate followed by the X2 coordinate and so on. For each scanline in the list, this routine will render a scanline from X1 to X2 (exclusive) at increasing Y coordinates. For scanlines where  $X2 < X1$ , the X1 and X2 coordinates will be swapped, and for scanlines where  $X1 = X2$ , the scanline will be skipped and nothing will be drawn.

### See Also

*DrawScan*, *DrawScanList*, *DrawColorPattScanList*, *Set8x8ColorPattern*, *Use8x8ColorPattern*

## DrawColorPattTrap

Draws a color patterned trapezoid.

### Declaration

```
void NAPI_GA_2DRenderFuncs::DrawColorPattTrap(
    GA_trap *trap)
```

### Prototype In

snap/graphics.h

### Parameters

*trap*                      Pointer to the *GA\_trap* structure describing the trapezoid

### Description

This function renders a color patterned, flat topped and bottomed trapezoid in the currently active mix and color pattern. The parameters for the trapezoid to be rendered are passed in the *GA\_trap* structure (note that all coordinates are in 16.16 fixed point format). This function will always be provided, and will be the workhorse function for rendering solid 2D polygons. After this function has been called, the driver will have updated the *y*, *x1* and *x2* variables in the *GA\_trap* structure to reflect the final values after scan converting the trapezoid. This ensures that the high level code can properly join up connected trapezoids to complete the rendering of a larger more complex polygon. Refer to *DrawTrap* for more information on the algorithm used to implement this drawing function.

### See Also

*DrawTrap*, *DrawColorPattTrap*, *DrawScanList*, *Set8x8ColorPattern*, *Use8x8ColorPattern*

## DrawEllipse

Draw a single pixel wide, outlined ellipse

### Declaration

```
void NAPI_GA_2DRenderFuncs::DrawEllipse(  
    N_int32 left,  
    N_int32 top,  
    N_int32 A,  
    N_int32 B)
```

### Prototype In

snap/graphics.h

### Parameters

<i>left</i>	Left coordinate for first pixel in the ellipse
<i>top</i>	Top coordinate for first pixel in the ellipse
<i>A</i>	Major axis dimension
<i>B</i>	Minor axis dimension

### Description

This function draws a single pixel wide, outlined ellipse using the currently active foreground color and mix.

### See Also

*ClipEllipse*

## DrawEllipseList

Draws a list of solid scanlines for a filled ellipse engine back end.

### Declaration

```
void NAPI_GA_2DRenderFuncs::DrawEllipseList(
    N_int32 y,
    N_int32 length,
    N_int32 height,
    N_int16 *scans)
```

### Prototype In

snap/graphics.h

### Parameters

<i>y</i>	Y coordinate for scanline
<i>length</i>	Number of scanlines in the list
<i>height</i>	Height of the ellipse minor axis
<i>scans</i>	Pointer to an array of scanline data

### Description

This function renders a list of solid scanlines starting at the specified location in the currently active color and mix. This function forms the back end of a fast filled ellipse rendering engine, but does not actually compute the scanlines in the list itself since the pixelisation rules are usually different for different device driver environments.

The scanline coordinates are passed as an array of 16-bit integer coordinates, packed with the LEFT coordinate followed by the RIGHT coordinate and so on for each scanline. For each scanline in the list, this routine will render a scanline from LEFT to RIGHT (exclusive) at increasing Y coordinates. The calling code must always guarantee that the LEFT coordinates will be less than the RIGHT coordinates, and that they will never be equal for each scanline.

This function will always be provided by accelerated drivers, and will be implemented with whatever hardware rendering function provides the fastest possible method of rendering scanlines with the installed hardware.

The algorithm used internally in the drivers to render the list of scanlines is similar to the following:

```
maxIndex = length-1;
for (i = 0, j = height; i < maxIndex; i++, j--, scans += 2) {
    SolidScan(i, scans[0], scans[1]);
    SolidScan(j, scans[0], scans[1]);
}
if (!(height & 1))
    SolidScan(i, scans[0], scans[0]);
```

### See Also

*DrawPattEllipseList, DrawColorPattEllipseList, DrawFatEllipseList, DrawPattFatEllipseList, DrawColorPattFatEllipseList*



## DrawFatEllipseList

Draws a list of solid scanlines for a fat ellipse engine back end.

### Declaration

```
void NAPI_GA_2DRenderFuncs::DrawFatEllipseList (
    N_int32 y,
    N_int32 length,
    N_int32 height,
    N_int16 *scans)
```

### Prototype In

snap/graphics.h

### Parameters

<i>y</i>	Y coordinate for scanline
<i>length</i>	Number of scanlines in the list
<i>height</i>	Height of the ellipse minor axis + pen height adjustment
<i>scans</i>	Pointer to an array of scanline data

### Description

This function renders a list of solid scanlines starting at the specified location in the currently active color and mix. This function forms the back end of a fast fat pen ellipse rendering engine, but does not actually compute the scanlines in the list itself since the pixelisation rules are usually different for different device driver environments.

The scanline coordinates are passed as an array of 16-bit integer coordinates, packed in multiples of 4 coordinates for a single scanline list. The first coordinate is the LEFTL coordinate, the second is the LEFTR, the third is the RIGHTL and the fourth is the RIGHTR coordinate. For each scanline in the list (each list defines two scanlines at the same Y coordinate), this routine will render a scanline from LEFT to RIGHT (exclusive) at increasing Y coordinates. The calling code must always guarantee that the LEFT coordinates will be less than the RIGHT coordinates, and that they will never be equal for each scanline.

This function will always be provided by accelerated drivers, and will be implemented with whatever hardware rendering function provides the fastest possible method of rendering scanlines with the installed hardware.

The algorithm used internally in the drivers to render the list of scanlines is similar to the following:

```
for (i = 0, j = height; i < length; i++, j--, scans += 4) {
    if (scans[LEFTR] < scans[RIGHTL]) {
        SolidScan(i, scans[LEFTL], scans[LEFTR]);
        SolidScan(i, scans[RIGHTL], scans[RIGHTR]);
        SolidScan(j, scans[LEFTL], scans[LEFTR]);
        SolidScan(j, scans[RIGHTL], scans[RIGHTR]);
    }
    else {
        SolidScan(i, scans[LEFTL], scans[RIGHTR]);
    }
}
```

```

        SolidScan(j, scans[LEFTL], scans[RIGHTR]);
    }
}
if ((height+1) & 1) {
    if (scans[LEFTR] < scans[RIGHTL]) {
        SolidScan(i, scans[LEFTL], scans[LEFTR]);
        SolidScan(i, scans[RIGHTL], scans[RIGHTR]);
    }
    else {
        SolidScan(i, scans[LEFTL], scans[RIGHTR]);
    }
}

```

**See Also**

*DrawEllipseList, DrawPattEllipseList, DrawColorPattEllipseList, DrawPattFatEllipseList, DrawColorPattFatEllipseList*

## DrawLineInt

Draws a solid, single pixel wide line with integer coordinates.

### Declaration

```
void NAPI_GA_2DRenderFuncs::DrawLineInt(
    N_int32 x1,
    N_int32 y1,
    N_int32 x2,
    N_int32 y2,
    N_int32 drawLast)
```

### Prototype In

snap/graphics.h

### Parameters

<i>x1</i>	X1 coordinate
<i>y1</i>	Y1 coordinate
<i>x2</i>	X2 coordinate
<i>y2</i>	Y2 coordinate
<i>drawLast</i>	1 to draw last pixel, 0 to skip it

### Description

This function renders a solid line at the specified location and the currently active color and mix. This routine will render a line from (x1,y1) to (x2,y2) inclusive. If the drawLast parameter is set, the last pixel in the line (x2,y2) will be drawn, otherwise it will be skipped. This feature allows multiple lines to be linked together as a polyline for CAD style operations while drawing in XOR mode (and is also required for compatibility with Microsoft Windows).

### See Also

*DrawBresenhamLine*, *DrawStippleLine*, *DrawStippleLineInt*, *DrawStyleLineInt*, *DrawClippedLineInt*

## DrawPattEllipseList

Draws a list of monochrome patterned scanlines for a filled ellipse engine back end.

### Declaration

```
void NAPI_GA_2DRenderFuncs::DrawPattEllipseList(
    N_int32 y,
    N_int32 length,
    N_int32 height,
    N_int16 *scans)
```

### Prototype In

snap/graphics.h

### Parameters

<i>y</i>	Y coordinate for scanline
<i>length</i>	Number of scanlines in the list
<i>height</i>	Height of the ellipse minor axis
<i>scans</i>	Pointer to an array of scanline data

### Description

This function renders a list of monochrome patterned scanlines starting at the specified location in the currently active colors, mix and monochrome pattern. This function forms the back end of a fast filled ellipse rendering engine, but does not actually compute the scanlines in the list itself since the pixelisation rules are usually different for different device driver environments.

The scanline coordinates are passed as an array of 16-bit integer coordinates, packed with the LEFT coordinate followed by the RIGHT coordinate and so on for each scanline. For each scanline in the list, this routine will render a scanline from LEFT to RIGHT (exclusive) at increasing Y coordinates. The calling code must always guarantee that the LEFT coordinates will be less than the RIGHT coordinates, and that they will never be equal for each scanline.

The algorithm used internally in the drivers to render the list of scanlines is similar to the following:

```
maxIndex = length-1;
for (i = 0, j = height; i < maxIndex; i++, j--, scans += 2) {
    PattScan(i, scans[0], scans[1]);
    PattScan(j, scans[0], scans[1]);
}
if (!(height & 1))
    PattScan(i, scans[0], scans[0]);
```

### See Also

*DrawEllipseList, DrawColorPattEllipseList, DrawFatEllipseList, DrawPattFatEllipseList, DrawColorPattFatEllipseList*

## DrawPattFatEllipseList

Draws a list of monochrome patterned scanlines for a fat ellipse engine back end.

### Declaration

```
void NAPI_GA_2DRenderFuncs::DrawPattFatEllipseList(
    N_int32 y,
    N_int32 length,
    N_int32 height,
    N_int16 *scans)
```

### Prototype In

snap/graphics.h

### Parameters

<i>y</i>	Y coordinate for scanline
<i>length</i>	Number of scanlines in the list
<i>height</i>	Height of the ellipse minor axis + pen height adjustment
<i>scans</i>	Pointer to an array of scanline data

### Description

This function renders a list of monochrome patterned scanlines starting at the specified location in the currently active colors, mix and monochrome pattern. This function forms the back end of a fast fat pen ellipse rendering engine, but does not actually compute the scanlines in the list itself since the pixelisation rules are usually different for different device driver environments.

The scanline coordinates are passed as an array of 16-bit integer coordinates, packed in multiples of 4 coordinates for a single scanline list. The first coordinate is the LEFTL coordinate, the second is the LEFTR, the third is the RIGHTL and the fourth is the RIGHTR coordinate. For each scanline in the list (each list defines two scanlines at the same Y coordinate), this routine will render a scanline from LEFT to RIGHT (exclusive) at increasing Y coordinates. The calling code must always guarantee that the LEFT coordinates will be less than the RIGHT coordinates, and that they will never be equal for each scanline.

The algorithm used internally in the drivers to render the list of scanlines is similar to the following:

```
for (i = 0, j = height; i < length; i++, j--, scans += 4) {
    if (scans[LEFTR] < scans[RIGHTL]) {
        PattScan(i, scans[LEFTL], scans[LEFTR]);
        PattScan(i, scans[RIGHTL], scans[RIGHTR]);
        PattScan(j, scans[LEFTL], scans[LEFTR]);
        PattScan(j, scans[RIGHTL], scans[RIGHTR]);
    }
    else {
        PattScan(i, scans[LEFTL], scans[RIGHTR]);
        PattScan(j, scans[LEFTL], scans[RIGHTR]);
    }
}
if ((height+1) & 1) {
```

```
if (scans[LEFTR] < scans[RIGHTL]) {  
    PattScan(i, scans[LEFTL], scans[LEFTR]);  
    PattScan(i, scans[RIGHTL], scans[RIGHTR]);  
}  
else {  
    PattScan(i, scans[LEFTL], scans[RIGHTR]);  
}  
}
```

**See Also**

*DrawEllipseList, DrawPattEllipseList, DrawColorPattEllipseList, DrawFatEllipseList,  
DrawColorPattFatEllipseList*

## DrawPattRect

Draws a monochrome pattern filled rectangle.

### Declaration

```
void NAPI_GA_2DRenderFuncs::DrawPattRect(
    N_int32 left,
    N_int32 top,
    N_int32 width,
    N_int32 height)
```

### Prototype In

snap/graphics.h

### Parameters

<i>left</i>	Left coordinate of the rectangle to draw
<i>top</i>	Top coordinate of the rectangle to draw
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines

### Description

This function renders a monochrome patterned rectangle at the specified location and in the currently active colors, mix and monochrome pattern. This routine will render a rectangle from (Left, Top) to (Left+Width-1, Height+Bottom-1) inclusive.

### See Also

*DrawRect, DrawColorPattRect, Set8x8MonoPattern, Use8x8MonoPattern*

## DrawPattScanList

Draws a list of monochrome patterned scanlines.

### Declaration

```
void NAPI_GA_2DRenderFuncs::DrawPattScanList(
    N_int32 y,
    N_int32 length,
    N_int16 *scans)
```

### Prototype In

snap/graphics.h

### Parameters

<i>y</i>	Y coordinate for scanline
<i>length</i>	Number of scanlines in the list
<i>scans</i>	Pointer to an array of scanline data

### Description

This function renders a list of monochrome patterned scanlines starting at the specified location in the currently active colors, mix and monochrome pattern. The scanline coordinates are passed as an array of 16-bit integer coordinates, packed with the X1 coordinate followed by the X2 coordinate and so on. For each scanline in the list, this routine will render a scanline from X1 to X2 (exclusive) at increasing Y coordinates. For scanlines where  $X2 < X1$ , the X1 and X2 coordinates will be swapped, and for scanlines where  $X1 = X2$ , the scanline will be skipped and nothing will be drawn.

### See Also

*DrawScan*, *DrawScanList*, *DrawColorPattScanList*, *Set8x8MonoPattern*, *Use8x8MonoPattern*



## DrawPattTrap

Draws a monochrome patterned trapezoid.

### Declaration

```
void NAPI GA_2DRenderFuncs::DrawPattTrap(
    GA_trap *trap)
```

### Prototype In

snap/graphics.h

### Parameters

*trap*                      Pointer to the *GA\_trap* structure describing the trapezoid

### Description

This function renders a monochrome patterned, flat topped and bottomed trapezoid in the currently active color, mix and monochrome pattern. The parameters for the trapezoid to be rendered are passed in the *GA\_trap* structure (note that all coordinates are in 16.16 fixed point format). This function will always be provided, and will be the workhorse function for rendering solid 2D polygons. After this function has been called, the driver will have updated the y, x1 and x2 variables in the *GA\_trap* structure to reflect the final values after scan converting the trapezoid. This ensures that the high level code can properly join up connected trapezoids to complete the rendering of a larger more complex polygon. Refer to *DrawTrap* for more information on the algorithm used to implement this drawing function.

### See Also

*DrawTrap*, *DrawColorPattTrap*, *DrawScanList*, *Set8x8MonoPattern*, *Use8x8MonoPattern*

## DrawRect

Draws a solid filled rectangle.

### Declaration

```
void NAPI_GA_2DRenderFuncs::DrawRect (
    N_int32 left,
    N_int32 top,
    N_int32 width,
    N_int32 height)
```

### Prototype In

snap/graphics.h

### Parameters

<i>left</i>	Left coordinate of the rectangle to draw
<i>top</i>	Top coordinate of the rectangle to draw
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines

### Description

This function renders a solid rectangle at the specified location and currently active color and mix. This routine will render a rectangle from (Left, Top) to (Left+Width-1, Height+Bottom-1) inclusive. This function will always be provided by accelerated drivers, and will be implemented with whatever hardware rendering function provides the fastest possible method of rendering rectangles with the installed hardware.

### See Also

*DrawPattRect*, *DrawColorPattRect*, *DrawRectLin*, *DrawRectExt*

## DrawRectExt

Draws a solid filled rectangle with specific color and mix

### Declaration

```
void NAPI_GA_2DRenderFuncs::DrawRectExt (
    N_int32 left,
    N_int32 top,
    N_int32 width,
    N_int32 height,
    GA_color color,
    N_int32 mix)
```

### Prototype In

snap/graphics.h

### Parameters

<i>left</i>	Left coordinate of the rectangle to draw
<i>top</i>	Top coordinate of the rectangle to draw
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>color</i>	Color to draw rectangle in
<i>mix</i>	Mix code to draw with ( <i>GA_mixCodesType</i> )

### Description

This function is identical to the *DrawRect* function, except that it also takes color and mix to draw the rectangle with. This function is intended primarily for high performance drawing of rectangles when changing the hardware state will be a performance burden.

### See Also

*DrawRect*, *DrawRectLin*

## DrawRectLin

Draws a solid filled rectangle with a linear source address.

### Declaration

```
void NAPI_GA_2DRenderFuncs::DrawRectLin(
    N_int32 dstOfs,
    N_int32 dstPitch,
    N_int32 left,
    N_int32 top,
    N_int32 width,
    N_int32 height,
    GA_color color,
    N_int32 mix)
```

### Prototype In

snap/graphics.h

### Parameters

<i>dstOfs</i>	Offset of destination rectangle in video memory
<i>dstPitch</i>	Pitch of destination rectangle in bytes
<i>left</i>	Left coordinate of the rectangle to draw
<i>top</i>	Top coordinate of the rectangle to draw
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>color</i>	Color to draw rectangle in
<i>mix</i>	Mix code to draw with ( <i>GA_mixCodesType</i> )

### Description

This function is identical to the *DrawRect* function, except that it also takes a destination linear offset, pitch, color and mix. This function is intended primarily for high performance DirectDraw compatibility for hardware that supports non-conforming linear memory addressing. For hardware that implements only (x,y) addressing, this function should not be implemented and DirectDraw will call the regular *DrawRect* function.

---

**Note:** *The value of dstOfs must be aligned to the boundary specified in the BitmapStartAlign member of the GA\_modeInfo structure, and the dstPitch value must be padded to multiples of the BitmapStridePad member of the GA\_modeInfo structure.*

---

### See Also

*DrawRect*, *DrawRectExt*

## DrawScanList

Draws a list of solid scanlines.

### Declaration

```
void NAPI_GA_2DRenderFuncs::DrawScanList(
    N_int32 y,
    N_int32 length,
    N_int16 *scans)
```

### Prototype In

snap/graphics.h

### Parameters

<i>y</i>	Y coordinate for scanline
<i>length</i>	Number of scanlines in the list
<i>scans</i>	Pointer to an array of scanline data

### Description

This function renders a list of solid scanlines starting at the specified location in the currently active color and mix. The scanline coordinates are passed as an array of 16-bit integer coordinates, packed with the X1 coordinate followed by the X2 coordinate and so on. For each scanline in the list, this routine will render a scanline from X1 to X2 (exclusive) at increasing Y coordinates. For scanlines where  $X2 < X1$ , the X1 and X2 coordinates will be swapped, and for scanlines where  $X1 = X2$ , the scanline will be skipped and nothing will be drawn. This function will always be provided by accelerated drivers, and will be implemented with whatever hardware rendering function provides the fastest possible method of rendering scanlines with the installed hardware. It is also one of the workhorse functions that will be used by high level rendering code for drawing non-polygonal solid shapes (ellipses, wedges, regions etc.).

### See Also

*DrawScan*, *DrawPattScanList*, *DrawColorPattScanList*

## DrawStippleLineInt

Draws a stippled, single pixel wide line with integer coordinates.

### Declaration

```
void NAPI_GA_2DRenderFuncs::DrawStippleLineInt(
    N_int32 x1,
    N_int32 y1,
    N_int32 x2,
    N_int32 y2,
    N_int32 drawLast,
    N_int32 transparent)
```

### Prototype In

snap/graphics.h

### Parameters

<i>x1</i>	X1 coordinate
<i>y1</i>	Y1 coordinate
<i>x2</i>	X2 coordinate
<i>y2</i>	Y2 coordinate
<i>drawLast</i>	1 to draw last pixel, 0 to skip it
<i>transparent</i>	1 if the line is transparent, 0 if opaque

### Description

This function renders a stippled line at the specified location and the currently active colors, mix and stipple pattern. This routine will render a line from (x1,y1) to (x2,y2) inclusive. If the drawLast parameter is set, the last pixel in the line (x2,y2) will be drawn, otherwise it will be skipped. This feature allows multiple lines to be linked together as a polyline for CAD style operations while drawing in XOR mode (and is also required for compatibility with Microsoft Windows).

If the transparent parameter is set to 1, where a bit is 0 in the stipple pattern the destination pixel remains untouched. If the transparent parameter is set to 0, where a bit is 0 in the stipple pattern the destination pixel is drawn in the background color. In all cases where a bit in the stipple pattern is 1, the pixel is drawn in the foreground color.

### See Also

*DrawBresenhamStippleLine, DrawLineInt, SetLineStipple, SetLineStippleCount*

## DrawStyleLineInt

Draws a OS/2 style styled, single pixel wide line with integer coordinates.

### Declaration

```
void NAPI_GA_2DRenderFuncs::DrawStyleLineInt(
    N_int32 x1,
    N_int32 y1,
    N_int32 x2,
    N_int32 y2,
    N_int32 drawLast,
    N_int32 transparent)
```

### Prototype In

snap/graphics.h

### Parameters

<i>x1</i>	X1 coordinate
<i>y1</i>	Y1 coordinate
<i>x2</i>	X2 coordinate
<i>y2</i>	Y2 coordinate
<i>drawLast</i>	1 to draw last pixel, 0 to skip it
<i>transparent</i>	1 if the line is transparent, 0 if opaque

### Description

This function renders an OS/2 style styled line at the specified location and the currently active colors, mix and style pattern. This routine will render a line from (x1,y1) to (x2,y2) inclusive. If the drawLast parameter is set, the last pixel in the line (x2,y2) will be drawn, otherwise it will be skipped. This feature allows multiple lines to be linked together as a polyline for CAD style operations while drawing in XOR mode (and is also required for compatibility with Microsoft Windows).

If the transparent parameter is set to 1, where a bit is 0 in the style pattern the destination pixel remains untouched. If the transparent parameter is set to 0, where a bit is 0 in the style pattern the destination pixel is drawn in the background color. In all cases where a bit in the style pattern is 1, the pixel is drawn in the foreground color.

### See Also

*DrawBresenhamStyleLine*, *DrawLineInt*, *SetLineStyle*

## DrawTrap

Draws a solid trapezoid.

### Declaration

```
void NAPI GA_2DRenderFuncs::DrawTrap(
    GA_trap *trap)
```

### Prototype In

snap/graphics.h

### Parameters

*trap*                      Pointer to the *GA\_trap* structure describing the trapezoid

### Description

This function renders a solid, flat topped and bottomed trapezoid in the currently active color and mix. The parameters for the trapezoid to be rendered are passed in the *GA\_trap* structure (note that all coordinates are in 16.16 fixed point format). This function will always be provided, and will be the workhorse function for rendering solid 2D polygons. After this function has been called, the driver will have updated the *y*, *x1* and *x2* variables in the *GA\_trap* structure to reflect the final values after scan converting the trapezoid. This ensures that the high level code can properly join up connected trapezoids to complete the rendering of a larger more complex polygon. The standard algorithm for implementing this in C is as follows (note that it handles edges that can cross within the trapezoid properly):

```
// Get input parameters into locals
N_int32 y = trap.y;
N_fix32 x1 = trap.x1;
N_fix32 x2 = trap.x2;

// Scan the trapezoid
while (trap.count-- > 0) {
    int ix1 = FIXROUND(x1);
    int ix2 = FIXROUND(x2);
    if (ix2 < ix1)
        SWAP(ix1, ix2);
    if (ix1 < ix2)
        scanLine(trap.y, ix1, ix2);
    x1 += slope1;
    x2 += slope2;
    y++;
}

// Update returned input parameters
trap.y = y;
trap.x1 = x1;
trap.x2 = x2;
```

### See Also

*DrawPattTrap*, *DrawColorPattTrap*, *DrawScanList*



## DstTransBlt

Copy a block of video memory to another location in video memory with destination transparency.

### Declaration

```
void NAPI GA_2DRenderFuncs::DstTransBlt(
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 width,
    N_int32 height,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 mix,
    GA_color transparent)
```

### Prototype In

snap/graphics.h

### Parameters

<i>srcLeft</i>	Left coordinate of the source rectangle to copy
<i>srcTop</i>	Top coordinate of the source rectangle to copy
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>mix</i>	Mix code for the copy ( <i>GA_mixCodesType</i> )
<i>transparent</i>	Transparent color value

### Description

This function copies a rectangular region of video memory from one location to another with destination transparency. This routine will copy a rectangular region of video memory from (srcLeft, srcTop, srcLeft+width-1, srcTop+height-1) to (dstLeft, dstTop) within video memory with the specified mix and with destination transparency. The mix code will be used to combine the source bitmap data with the pixels in the destination bitmap. The transparent color passed will be used to *mask out* pixels in the destination bitmap from being written. Where a pixel in the destination bitmap matches the transparent color, the pixel will be written to the destination bitmap. The results of this function are undefined if the source and destination rectangles overlap.

---

**Note:** *Although you can achieve the same effect of this routine using the generic BitBltFx function, this function is provided separately as it is usually a workhorse function for sprite based game applications and needs to be as efficient as possible.*

---

### See Also

*DstTransBltLin, DstTransBltSys, DstTransBltBM, DstTransBlt, BitBlt, BitBltFx*

## DstTransBltBM

Copy a block of system memory to a location in video memory with Bus Mastering and destination transparency.

### Declaration

```
void NAPI_GA_2DRenderFuncs::DstTransBltBM(
    void *srcAddr,
    N_int32 srcPhysAddr,
    N_int32 srcPitch,
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 width,
    N_int32 height,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 mix,
    GA_color transparent)
```

### Prototype In

snap/graphics.h

### Parameters

<i>srcAddr</i>	Address of source bitmap in system memory
<i>srcPhysAddr</i>	Physical address of source bitmap in system memory
<i>srcPitch</i>	Pitch of source bitmap in bytes
<i>srcLeft</i>	Left coordinate within source bitmap to copy
<i>srcTop</i>	Top coordinate within source bitmap to copy
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>mix</i>	Mix code for the copy ( <i>GA_mixCodesType</i> )
<i>transparent</i>	Transparent color value

### Description

This routine will copy a bitmap from system memory with a physical starting address of *srcPhysAddr* to the destination rectangle (*dstLeft*, *dstTop*, *dstLeft*+*width*-1, *dstTop*+*height*-1) with the specified mix and with destination transparency. The mix code will be used to combine the source bitmap data with the pixels in the destination bitmap. The transparent color passed will be used to *mask out* pixels in the destination bitmap from being written. Where a pixel in the destination bitmap matches the transparent color, the pixel will be written to the destination bitmap. The *srcPhysAddr* value points to the start of the bitmap data in system memory as a *physical* memory address, not a linear memory address that the application software normally deals with. It is up to the calling application to use the necessary OS services to allocate a block of contiguous physical memory for the bitmap data, and to obtain the physical memory address to be passed into this function.

This version is different to the *DstTransBltSys* function in that the bitmap data is copied using Bus Mastering by the graphics accelerator, which allows this function to return before the copy has completed and the accelerator will complete the copy in the background with a DMA Bus Master operation. If this hardware supports Bus Mastering and this function is available, it will usually be the fastest method to copy a block of system memory to video memory.

Note that the *srcLeft* and *srcTop* coordinates define an offset within the source bitmap to be copied, so it will copy only a portion of the memory bitmap.

---

**Note:** *Although you can achieve the same effect of this routine using the generic *BitBltExBM* function, this function is provided separately as it is usually a workhorse function for sprite based game applications and needs to be as efficient as possible.*

---

**See Also**

*DstTransBlt, DstTransBltLin, DstTransBltSys, DstTransBlt, BitBlt, BitBltEx*

## DstTransBltLin

Copy a linear block of video memory to another location in video memory with destination transparency.

### Declaration

```
void NAPI GA_2DRenderFuncs::DstTransBltLin(
    N_int32 srcOfs,
    N_int32 srcPitch,
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 width,
    N_int32 height,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 mix,
    GA_color transparent)
```

### Prototype In

snap/graphics.h

### Parameters

<i>srcOfs</i>	Offset of source bitmap in video memory
<i>srcPitch</i>	Pitch of source bitmap in bytes
<i>srcLeft</i>	Left coordinate within source bitmap to copy
<i>srcTop</i>	Top coordinate within source bitmap to copy
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>mix</i>	Mix code for the copy ( <i>GA_mixCodesType</i> )
<i>transparent</i>	Transparent color value

### Description

This routine will copy a linear region of video memory from *srcOfs* from the start of video memory to the destination rectangle (*dstLeft*, *dstTop*, *dstLeft*+*width*-1, *dstTop*+*height*-1) with the specified mix and with destination transparency. The mix code will be used to combine the source bitmap data with the pixels in the destination bitmap. The transparent color passed will be used to *mask out* pixels in the destination bitmap from being written. Where a pixel in the destination bitmap matches the transparent color, the pixel will be written to the destination bitmap. Note that the value of *srcOfs* must be aligned to the boundary specified in the *BitmapStartAlign* member of the *GA\_modeInfo* structure, and the *srcPitch* value must be padded to multiples of the *BitmapStridePad* member of the *GA\_modeInfo* structure. The results of this routine are undefined if the video memory regions overlap.

Note that the *srcLeft* and *srcTop* coordinates define an offset within the source bitmap to be copied, so it will copy only a portion of an offscreen bitmap. This is useful for storing multiple images in a single offscreen bitmap, or for handling the case of software

clipping offscreen bitmaps if the destination lies outside of the software clip rectangle for the destination buffer.

This version is different to the standard *BitBlt* function in that the source bitmap to be copied can be non-conforming, and can have a different logical scanline width to the destination bitmap. This allows the bitmaps to be stored contiguously in offscreen video memory, rather than requiring the offscreen video memory to be divided up into rectangular regions, resulting in more efficient use of available offscreen memory for bitmap storage.

---

**Note:** *The value of `srcOfs` must be aligned to the boundary specified in the `BitmapStartAlign` member of the `GA_modeInfo` structure, and the `dstPitch` value must be padded to multiples of the `BitmapStridePad` member of the `GA_modeInfo` structure.*

**Note:** *Although you can achieve the same effect of this routine using the generic `BitBltExLin` function, this function is provided separately as it is usually a workhorse function for sprite based game applications and needs to be as efficient as possible.*

---

#### **See Also**

*`DstTransBlt`, `DstTransBltSys`, `DstTransBltBM`, `DstTransBlt`, `BitBlt`, `BitBltEx`*

## DstTransBltSys

Copy a block of system memory to a location in video memory with destination transparency.

### Declaration

```
void NAPI_GA_2DRenderFuncs::DstTransBltSys(
    void *srcAddr,
    N_int32 srcPitch,
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 width,
    N_int32 height,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 mix,
    GA_color transparent,
    N_int32 flipY)
```

### Prototype In

snap/graphics.h

### Parameters

<i>srcAddr</i>	Address of source bitmap in system memory
<i>srcPitch</i>	Pitch of source bitmap in bytes
<i>srcLeft</i>	Left coordinate within source bitmap to copy
<i>srcTop</i>	Top coordinate within source bitmap to copy
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>mix</i>	Mix code for the copy ( <i>GA_mixCodesType</i> )
<i>transparent</i>	Transparent color value
<i>flipY</i>	True if the image should be flipped vertically

### Description

This routine will copy a bitmap from system memory with a starting address of *srcAddr* to the destination rectangle (*dstLeft*, *dstTop*, *dstLeft*+*width*-1, *dstTop*+*height*-1) with the specified mix and with destination transparency. The mix code will be used to combine the source bitmap data with the pixels in the destination bitmap. The transparent color passed will be used to *mask out* pixels in the destination bitmap from being written. Where a pixel in the destination bitmap matches the transparent color, the pixel will be written to the destination bitmap.

Note that the *srcLeft* and *srcTop* coordinates define an offset within the source bitmap to be copied, so it will copy only a portion of the memory bitmap.

---

**Note:** *Although you can achieve the same effect of this routine using the generic `BitBltFxFxSys` function, this function is provided separately as it is usually a workhorse function for sprite based game applications and needs to be as efficient as possible.*

---

**See Also**

*DstTransBlt, DstTransBltLin, DstTransBltBM, DstTransBlt, BitBlt, BitBltEx*

## GetBitmapBM

Copy a block of video memory to a location in system memory with Bus Mastering.

### Declaration

```
void NAPI_GA_2DRenderFuncs::GetBitmapBM(
    void *dstAddr,
    N_int32 dstPhysAddr,
    N_int32 dstPitch,
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 width,
    N_int32 height,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 mix)
```

### Prototype In

snap/graphics.h

### Parameters

<i>dstAddr</i>	Address of destination bitmap in system memory
<i>dstPhysAddr</i>	Physical address of destination bitmap in system memory
<i>dstPitch</i>	Pitch of destination bitmap in bytes
<i>srcLeft</i>	Left coordinate within source bitmap to copy
<i>srcTop</i>	Top coordinate within source bitmap to copy
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>mix</i>	Mix code for the copy ( <i>GA_mixCodesType</i> )

### Description

This routine will copy a bitmap from video memory to system memory with a starting address of *dstPhysAddr* from the source rectangle (*srcLeft*, *srcTop*, *srcLeft*+*width*-1, *srcTop*+*height*-1). The *dstPhysAddr* value points to the start of the destination bitmap data in system memory as a *physical* memory address, not a linear memory address that the application software normally deals with. It is up to the calling application to use the necessary OS services to allocate a block of contiguous physical memory for the bitmap data, and to obtain the physical memory address to be passed into this function.

---

**Note:** *This function is only implemented for hardware that can do bus master reads over the PCI bus, and may be significantly faster than code that simply does direct reads over the PCI bus. Note that this function may return before the bus master operation has completed, and the application code should call the WaitTillIdle function to determine when the bus master operation has completed before using the data in the destination bitmap buffer.*

---

### See Also

*BitBlt*, *WaitTillIdle*, *GetBitmapSys*



## GetBitmapSys

Copy a block of video memory to a location in system memory.

### Declaration

```
void NAPI_GA_2DRenderFuncs::GetBitmapSys(
    void *dstAddr,
    N_int32 dstPitch,
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 width,
    N_int32 height,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 mix)
```

### Prototype In

snap/graphics.h

### Parameters

<i>dstAddr</i>	Address of destination bitmap in system memory
<i>dstPitch</i>	Pitch of destination bitmap in bytes
<i>srcLeft</i>	Left coordinate within source bitmap to copy
<i>srcTop</i>	Top coordinate within source bitmap to copy
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>mix</i>	Mix code for the copy ( <i>GA_mixCodesType</i> )

### Description

This routine will copy a bitmap from video memory to system memory with a starting address of *dstAddr* from the source rectangle (*srcLeft*, *srcTop*, *srcLeft*+*width*-1, *srcTop*+*height*-1).

### See Also

*BitBlt*, *WaitTillIdle*, *GetBitmapBM*

## GetPixel

Reads a pixel value from the framebuffer

### Declaration

```
GA_color NAPI GA_2DRenderFuncs::GetPixel(  
    N_int32 x,  
    N_int32 y)
```

### Prototype In

snap/graphics.h

### Parameters

<i>x</i>	X coordinate to read pixel value from
<i>y</i>	Y coordinate to read pixel value from

### Description

This function reads the color of a single pixel from the framebuffer.

### See Also

*PutPixel*

## PutMonoImageLSBBM

Draws a monochrome bitmap stored in system memory with bus mastering

### Declaration

```
void NAPI_GA_2DRenderFuncs::PutMonoImageLSBBM(
    N_int32 x,
    N_int32 y,
    N_int32 width,
    N_int32 height,
    N_int32 byteWidth,
    N_uint8 *image,
    N_int32 imagePhysAddr,
    N_int32 transparent)
```

### Prototype In

snap/graphics.h

### Parameters

<i>x</i>	Destination X coordinate to draw the bitmap at
<i>y</i>	Destination Y coordinate to draw the bitmap at
<i>width</i>	Width of the bitmap in pixels
<i>height</i>	Height of the bitmap in pixels
<i>byteWidth</i>	Width of the bitmap in bytes
<i>image</i>	Pointer to the bitmap image data to draw
<i>imagePhysAddr</i>	Physical address of bitmap image data in system memory
<i>transparent</i>	1 for transparent, 0 for opaque

### Description

This function is identical to the *PutMonoImageMSBBM* function, except that it processes the bitmap data in LSB fashion. This means that the first pixel drawn corresponds to bit 0 of the first byte. The second pixel is bit 1 of the first byte, ..., the 8th pixel is bit 0 of the second byte etc. Both LSB and MSB versions are provided for performance.

### See Also

*PutMonoImageLSBSys*, *PutMonoImageLSBBM*, *PutMonoImageMSBSys*,  
*PutMonoImageMSBLin*, *PutMonoImageMSBBM*, *ClipMonoImageLSBSys*,  
*ClipMonoImageLSBLin*, *ClipMonoImageLSBBM*, *ClipMonoImageMSBSys*,  
*ClipMonoImageMSBLin*, *ClipMonoImageMSBBM*

## PutMonoImageLSBLin

Draws a monochrome bitmap stored in offscreen video memory

### Declaration

```
void NAPI_GA_2DRenderFuncs::PutMonoImageLSBLin(
    N_int32 x,
    N_int32 y,
    N_int32 width,
    N_int32 height,
    N_int32 byteWidth,
    N_int32 imageOfs,
    N_int32 transparent)
```

### Prototype In

snap/graphics.h

### Parameters

<i>x</i>	Destination X coordinate to draw the bitmap at
<i>y</i>	Destination Y coordinate to draw the bitmap at
<i>width</i>	Width of the bitmap in pixels
<i>height</i>	Height of the bitmap in pixels
<i>byteWidth</i>	Width of the bitmap in bytes
<i>imageOfs</i>	Offset of bitmap image data in video memory (byte address)
<i>transparent</i>	1 for transparent, 0 for opaque

### Description

This function is identical to the *PutMonoImageMSBLin* function, except that it processes the bitmap data in LSB fashion. This means that the first pixel drawn corresponds to bit 0 of the first byte. The second pixel is bit 1 of the first byte, ..., the 8th pixel is bit 0 of the second byte etc. Both LSB and MSB versions are provided for performance.

---

**Note:** *The value of imageOfs must be aligned to the boundary specified in the MonoBitmapStartAlign member of the GA\_modeInfo structure, and the byteWidth value must be padded to multiples of the MonoBitmapStridePad member of the GA\_modeInfo structure.*

---

### See Also

*PutMonoImageLSBSys, PutMonoImageLSBBM, PutMonoImageMSBSys, PutMonoImageMSBLin, PutMonoImageMSBBM, ClipMonoImageLSBSys, ClipMonoImageLSBLin, ClipMonoImageLSBBM, ClipMonoImageMSBSys, ClipMonoImageMSBLin, ClipMonoImageMSBBM*

## PutMonoImageLSBSys

Draws a monochrome bitmap stored in system memory

### Declaration

```
void NAPI_GA_2DRenderFuncs::PutMonoImageLSBSys (
    N_int32 x,
    N_int32 y,
    N_int32 width,
    N_int32 height,
    N_int32 byteWidth,
    N_uint8 *image,
    N_int32 transparent)
```

### Prototype In

snap/graphics.h

### Parameters

<i>x</i>	Destination X coordinate to draw the bitmap at
<i>y</i>	Destination Y coordinate to draw the bitmap at
<i>width</i>	Width of the bitmap in pixels
<i>height</i>	Height of the bitmap in pixels
<i>byteWidth</i>	Width of the bitmap in bytes
<i>image</i>	Pointer to the bitmap image data to draw
<i>transparent</i>	1 for transparent, 0 for opaque

### Description

This function is identical to the *PutMonoImageMSBSys* function, except that it processes the bitmap data in LSB fashion. This means that the first pixel drawn corresponds to bit 0 of the first byte. The second pixel is bit 1 of the first byte, ..., the 8th pixel is bit 0 of the second byte etc. Both LSB and MSB versions are provided for performance.

### See Also

*PutMonoImageLSBLin*, *PutMonoImageLSBBM*, *PutMonoImageMSBSys*,  
*PutMonoImageMSBLin*, *PutMonoImageMSBBM*, *ClipMonoImageLSBSys*,  
*ClipMonoImageLSBLin*, *ClipMonoImageLSBBM*, *ClipMonoImageMSBSys*,  
*ClipMonoImageMSBLin*, *ClipMonoImageMSBBM*

## PutMonoImageMSBBM

Draws a monochrome bitmap stored in system memory with bus mastering

### Declaration

```
void NAPI_GA_2DRenderFuncs::PutMonoImageMSBBM(
    N_int32 x,
    N_int32 y,
    N_int32 width,
    N_int32 height,
    N_int32 byteWidth,
    N_uint8 *image,
    N_int32 imagePhysAddr,
    N_int32 transparent)
```

### Prototype In

snap/graphics.h

### Parameters

<i>x</i>	Destination X coordinate to draw the bitmap at
<i>y</i>	Destination Y coordinate to draw the bitmap at
<i>width</i>	Width of the bitmap in pixels
<i>height</i>	Height of the bitmap in pixels
<i>byteWidth</i>	Width of the bitmap in bytes
<i>image</i>	Pointer to the bitmap image data to draw
<i>imagePhysAddr</i>	Physical address of bitmap image data in system memory
<i>transparent</i>	1 for transparent, 0 for opaque

### Description

This function copies a monochrome bitmap image from a system memory buffer to video memory using the hardware accelerator, which is used for fast bitmap masking and font rendering operations. The bitmap is rendered in the specified colors using the currently active mix. This function is identical to *PutMonoImageMSBSys*, except that the bitmap data transferred to video memory using a bus master DMA operation, and the *imagePhysAddr* is the physical memory address of the image in system memory (and the bitmap data *must* be physically contiguous in memory).

### See Also

*PutMonoImageMSBSys*, *PutMonoImageMSBBM*, *PutMonoImageMSBSys*,  
*PutMonoImageLSBLin*, *PutMonoImageLSBBM*, *ClipMonoImageLSBSys*,  
*ClipMonoImageLSBLin*, *ClipMonoImageLSBBM*, *ClipMonoImageMSBSys*,  
*ClipMonoImageMSBLin*, *ClipMonoImageMSBBM*

## PutMonoImageMSBLin

Draws a monochrome bitmap stored in offscreen video memory

### Declaration

```
void NAPI_GA_2DRenderFuncs::PutMonoImageMSBLin(
    N_int32 x,
    N_int32 y,
    N_int32 width,
    N_int32 height,
    N_int32 byteWidth,
    N_int32 imageOfs,
    N_int32 transparent)
```

### Prototype In

snap/graphics.h

### Parameters

<i>x</i>	Destination X coordinate to draw the bitmap at
<i>y</i>	Destination Y coordinate to draw the bitmap at
<i>width</i>	Width of the bitmap in pixels
<i>height</i>	Height of the bitmap in pixels
<i>byteWidth</i>	Width of the bitmap in bytes
<i>imageOfs</i>	Offset of bitmap image data in video memory (byte address)
<i>transparent</i>	1 for transparent, 0 for opaque

### Description

This function copies a monochrome bitmap image from a video memory buffer to video memory using the hardware accelerator, which is used for fast bitmap masking and font rendering operations. The bitmap is rendered in the specified colors using the currently active mix. This function is identical to *PutMonoImageMSBSys*, except that the bitmap data is taken from a packed bitmap image in video memory, with the *imageOfs* parameter pointing to the start of the bitmap in video memory.

---

**Note:** *The value of imageOfs must be aligned to the boundary specified in the MonoBitmapStartAlign member of the GA\_modeInfo structure, and the byteWidth value must be padded to multiples of the MonoBitmapStridePad member of the GA\_modeInfo structure.*

---

### See Also

*PutMonoImageMSBSys, PutMonoImageMSBBM, PutMonoImageMSBSys, PutMonoImageLSBLin, PutMonoImageLSBBM, ClipMonoImageLSBSys, ClipMonoImageLSBLin, ClipMonoImageLSBBM, ClipMonoImageMSBSys, ClipMonoImageMSBLin, ClipMonoImageMSBBM*

## PutMonoImageMSBSys

Draws a monochrome bitmap stored in system memory

### Declaration

```
void NAPI_GA_2DRenderFuncs::PutMonoImageMSBSys (
    N_int32 x,
    N_int32 y,
    N_int32 width,
    N_int32 height,
    N_int32 byteWidth,
    N_uint8 *image,
    N_int32 transparent)
```

### Prototype In

snap/graphics.h

### Parameters

<i>x</i>	Destination X coordinate to draw the bitmap at
<i>y</i>	Destination Y coordinate to draw the bitmap at
<i>width</i>	Width of the bitmap in pixels
<i>height</i>	Height of the bitmap in pixels
<i>byteWidth</i>	Width of the bitmap in bytes
<i>image</i>	Pointer to the bitmap image data to draw
<i>transparent</i>	1 for transparent, 0 for opaque

### Description

This function copies a monochrome bitmap image from a system memory buffer to video memory using the hardware accelerator, which is used for fast bitmap masking and font rendering operations. The bitmap is rendered using the currently active colors and mix.

The *x* and *y* parameters define the destination coordinate for the image, and the *width* and *height* parameters define the dimensions of the monochrome image to be displayed. The *byteWidth* parameter defines the width of the bitmap image in bytes, and *must* be equal to the value of  $(width + 7) / 8$ . Hence the *width* parameter is used to clip off unwanted pixels on the right hand edge of the bitmap, but it cannot clip off more than a single bytes worth of pixels. The image pointer points to the start of the monochrome image in system memory and is byte packed.

If the *transparent* parameter is set to 1, where a bit is 0 in the bitmap image the destination pixel remains untouched. If the *transparent* parameter is set to 0, where a bit is 0 in the bitmap image the destination pixel is drawn in the background color. In all cases where a bit in the bitmap image is 1, the pixel is drawn in the foreground color.

---

**Note:** *This function processes the bitmap data in MSB fashion, in that the first pixel drawn corresponds to bit 7 of the first byte. The second pixel is bit 6 of the first byte, ..., the 8th pixel is bit 7 of the second byte etc. Both LSB and MSB versions are provided for performance.*

---



---

**Note:** *Both the MSB and LSB versions should only be implemented if the hardware supports both bitmap processing modes. If the hardware only supports one mode, only that mode must be implemented so that the higher level code can use the most optimal method of bit-swizzling the bitmap data before sending it to the driver.*

---

**See Also**

*PutMonoImageMSBLin, PutMonoImageMSBBM, PutMonoImageMSBSys,  
PutMonoImageLSBLin, PutMonoImageLSBBM, ClipMonoImageLSBSys,  
ClipMonoImageLSBLin, ClipMonoImageLSBBM, ClipMonoImageMSBSys,  
ClipMonoImageMSBLin, ClipMonoImageMSBBM*

## PutPixel

Draws a pixel value into the framebuffer

### Declaration

```
void NAPI_GA_2DRenderFuncs::PutPixel(  
    N_int32 x,  
    N_int32 y)
```

### Prototype In

snap/graphics.h

### Parameters

<i>x</i>	X coordinate to draw pixel at
<i>y</i>	Y coordinate to draw pixel at

### Description

This function draws a single pixel into the framebuffer using the current active foreground color and mix.

### See Also

*GetPixel*

## SrcTransBlt

Copy a block of video memory to another location in video memory with source transparency.

### Declaration

```
void NAPI GA_2DRenderFuncs::SrcTransBlt(
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 width,
    N_int32 height,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 mix,
    GA_color transparent)
```

### Prototype In

snap/graphics.h

### Parameters

<i>srcLeft</i>	Left coordinate of the source rectangle to copy
<i>srcTop</i>	Top coordinate of the source rectangle to copy
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>mix</i>	Mix code for the copy ( <i>GA_mixCodesType</i> )
<i>transparent</i>	Transparent color value

### Description

This function copies a rectangular region of video memory from one location to another with source transparency. This routine will copy a rectangular region of video memory from (srcLeft, srcTop, srcLeft+width-1, srcTop+height-1) to (dstLeft, dstTop) within video memory with the specified mix and with source transparency. The mix code will be used to combine the source bitmap data with the pixels in the destination bitmap. The transparent color passed will be used to *mask out* pixels in the source bitmap from being written to the destination area. Where a pixel in the source bitmap matches the transparent color, the pixel will not be written to the destination bitmap. The results of this function are undefined if the source and destination rectangles overlap.

---

**Note:** *Although you can achieve the same effect of this routine using the generic BitBltFx function, this function is provided separately as it is usually a workhorse function for sprite based game applications and needs to be as efficient as possible.*

---

### See Also

*SrcTransBltLin, SrcTransBltSys, SrcTransBltBM, DstTransBlt, BitBlt, BitBltFx*

## SrcTransBltBM

Copy a block of system memory to a location in video memory with Bus Mastering and source transparency.

### Declaration

```
void NAPI_GA_2DRenderFuncs::SrcTransBltBM(
    void *srcAddr,
    N_int32 srcPhysAddr,
    N_int32 srcPitch,
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 width,
    N_int32 height,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 mix,
    GA_color transparent)
```

### Prototype In

snap/graphics.h

### Parameters

<i>srcAddr</i>	Address of source bitmap in system memory
<i>srcPhysAddr</i>	Physical address of source bitmap in system memory
<i>srcPitch</i>	Pitch of source bitmap in bytes
<i>srcLeft</i>	Left coordinate within source bitmap to copy
<i>srcTop</i>	Top coordinate within source bitmap to copy
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>mix</i>	Mix code for the copy ( <i>GA_mixCodesType</i> )
<i>transparent</i>	Transparent color value

### Description

This routine will copy a bitmap from system memory with a physical starting address of *srcPhysAddr* to the destination rectangle (*dstLeft*, *dstTop*, *dstLeft*+*width*-1, *dstTop*+*height*-1) with the specified mix and with source transparency. The mix code will be used to combine the source bitmap data with the pixels in the destination bitmap. The transparent color passed will be used to *mask out* pixels in the source bitmap from being written to the destination area. Where a pixel in the source bitmap matches the transparent color, the pixel will not be written to the destination bitmap. The *srcPhysAddr* value points to the start of the bitmap data in system memory as a *physical* memory address, not a linear memory address that the application software normally deals with. It is up to the calling application to use the necessary OS services to allocate a block of contiguous physical memory for the bitmap data, and to obtain the physical memory address to be passed into this function.

This version is different to the *SrcTransBltSys* function in that the bitmap data is copied using Bus Mastering by the graphics accelerator, which allows this function to return before the copy has completed and the accelerator will complete the copy in the background with a DMA Bus Master operation. If this hardware supports Bus Mastering and this function is available, it will usually be the fastest method to copy a block of system memory to video memory.

Note that the *srcLeft* and *srcTop* coordinates define an offset within the source bitmap to be copied, so it will copy only a portion of the memory bitmap.

---

**Note:** *Although you can achieve the same effect of this routine using the generic *BitBltFxBM* function, this function is provided separately as it is usually a workhorse function for sprite based game applications and needs to be as efficient as possible.*

---

**See Also**

*SrcTransBlt, SrcTransBltLin, SrcTransBltSys, DstTransBlt, BitBlt, BitBltFx*

## SrcTransBlitLin

Copy a linear block of video memory to another location in video memory with source transparency.

### Declaration

```
void NAPI_GA_2DRenderFuncs::SrcTransBlitLin(
    N_int32 srcOfs,
    N_int32 srcPitch,
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 width,
    N_int32 height,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 mix,
    GA_color transparent)
```

### Prototype In

snap/graphics.h

### Parameters

<i>srcOfs</i>	Offset of source bitmap in video memory
<i>srcPitch</i>	Pitch of source bitmap in bytes
<i>srcLeft</i>	Left coordinate within source bitmap to copy
<i>srcTop</i>	Top coordinate within source bitmap to copy
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>mix</i>	Mix code for the copy ( <i>GA_mixCodesType</i> )
<i>transparent</i>	Transparent color value

### Description

This routine will copy a linear region of video memory from *srcOfs* from the start of video memory to the destination rectangle (*dstLeft*, *dstTop*, *dstLeft*+*width*-1, *dstTop*+*height*-1) with the specified mix and with source transparency. The mix code will be used to combine the source bitmap data with the pixels in the destination bitmap. The transparent color passed will be used to *mask out* pixels in the source bitmap from being written to the destination area. Where a pixel in the source bitmap matches the transparent color, the pixel will not be written to the destination bitmap. Note that the value of *srcOfs* must be aligned to the boundary specified in the *BitmapStartAlign* member of the *GA\_modeInfo* structure, and the *srcPitch* value must be padded to multiples of the *BitmapStridePad* member of the *GA\_modeInfo* structure. The results of this routine are undefined if the video memory regions overlap.

Note that the *srcLeft* and *srcTop* coordinates define an offset within the source bitmap to be copied, so it will copy only a portion of an offscreen bitmap. This is useful for storing multiple images in a single offscreen bitmap, or for handling the case of software

clipping offscreen bitmaps if the destination lies outside of the software clip rectangle for the destination buffer.

This version is different to the standard *BitBlt* function in that the source bitmap to be copied can be non-conforming, and can have a different logical scanline width to the destination bitmap. This allows the bitmaps to be stored contiguously in offscreen video memory, rather than requiring the offscreen video memory to be divided up into rectangular regions, resulting in more efficient use of available offscreen memory for bitmap storage.

---

**Note:** *The value of `srcOfs` must be aligned to the boundary specified in the `BitmapStartAlign` member of the `GA_modeInfo` structure, and the `dstPitch` value must be padded to multiples of the `BitmapStridePad` member of the `GA_modeInfo` structure.*

**Note:** *Although you can achieve the same effect of this routine using the generic `BitBltFxFxLin` function, this function is provided separately as it is usually a workhorse function for sprite based game applications and needs to be as efficient as possible.*

---

#### **See Also**

*SrcTransBlt, SrcTransBltSys, SrcTransBltBM, DstTransBlt, BitBlt, BitBltFx*

## SrcTransBltSys

Copy a block of system memory to a location in video memory with source transparency.

### Declaration

```
void NAPI_GA_2DRenderFuncs::SrcTransBltSys(
    void *srcAddr,
    N_int32 srcPitch,
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 width,
    N_int32 height,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 mix,
    GA_color transparent,
    N_int32 flipY)
```

### Prototype In

snap/graphics.h

### Parameters

<i>srcAddr</i>	Address of source bitmap in system memory
<i>srcPitch</i>	Pitch of source bitmap in bytes
<i>srcLeft</i>	Left coordinate within source bitmap to copy
<i>srcTop</i>	Top coordinate within source bitmap to copy
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>mix</i>	Mix code for the copy ( <i>GA_mixCodesType</i> )
<i>transparent</i>	Transparent color value
<i>flipY</i>	True if the image should be flipped vertically

### Description

This routine will copy a bitmap from system memory with a starting address of *srcAddr* to the destination rectangle (*dstLeft*, *dstTop*, *dstLeft*+*width*-1, *dstTop*+*height*-1) with the specified mix and with source transparency. The mix code will be used to combine the source bitmap data with the pixels in the destination bitmap. The transparent color passed will be used to /mask out/ pixels in the source bitmap from being written to the destination area. Where a pixel in the source bitmap matches the transparent color, the pixel will not be written to the destination bitmap.

Note that the *srcLeft* and *srcTop* coordinates define an offset within the source bitmap to be copied, so it will copy only a portion of the memory bitmap.

---

**Note:** *Although you can achieve the same effect of this routine using the generic `BitBltFxFxSys` function, this function is provided separately as it is usually a workhorse function for sprite based game applications and needs to be as efficient as possible.*

---



**See Also**

*SrcTransBlt, SrcTransBltLin, SrcTransBltBM, DstTransBlt, BitBlt, BitBltEx*

## StretchBlt

Copy a block of video memory to another location in video memory with stretching or shrinking.

### Declaration

```
void NAPI GA_2DRenderFuncs::StretchBlt(
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 srcWidth,
    N_int32 srcHeight,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 dstWidth,
    N_int32 dstHeight,
    N_int32 doClip,
    N_int32 clipLeft,
    N_int32 clipTop,
    N_int32 clipRight,
    N_int32 clipBottom,
    N_int32 mix)
```

### Prototype In

snap/graphics.h

### Parameters

<i>srcLeft</i>	Left coordinate of the source rectangle to copy
<i>srcTop</i>	Top coordinate of the source rectangle to copy
<i>srcWidth</i>	Width of the source rectangle in pixels
<i>srcHeight</i>	Height of the source rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>dstWidth</i>	Width of the destination rectangle in pixels
<i>dstHeight</i>	Height of the destination rectangle in scanlines
<i>doClip</i>	True if the blit should be clipped, false if not
<i>clipLeft</i>	Left coordinate for clip rectangle (inclusive)
<i>clipTop</i>	Top coordinate for clip rectangle (inclusive)
<i>clipRight</i>	Right coordinate for clip rectangle (exclusive)
<i>clipBottom</i>	Bottom coordinate for clip rectangle (exclusive)
<i>mix</i>	Mix code for the copy ( <i>GA_mixCodesType</i> )

### Description

This function copies a rectangular region of video memory from one location to another with either stretching or shrinking. This routine will copy the rectangular region of video memory from (srcLeft, srcTop, srcLeft+srcWidth-1, srcTop+srcHeight-1) to (dstLeft, dstTop, dstLeft+dstWidth-1, dstTop+dstHeight-1) within video memory. Note that the source and destination rectangle dimensions may be different in, which is the case for doing a copy with bitmap stretching or shrinking. The results of this routine are undefined if the video memory regions overlap.

If the `doClip` parameter is true, then the output of the stretch function will be clipped against the passed in destination clip rectangle.

**See Also**

*StretchBltLin, StretchBltSys, StretchBltBM, SrcTransBlt, DstTransBlt, BitBlt*

## StretchBltBM

Copy a block of system memory to another location in video memory with stretching or shrinking and bus mastering.

### Declaration

```
void NAPI_GA_2DRenderFuncs::StretchBltBM(
    void *srcAddr,
    N_int32 srcPhysAddr,
    N_int32 srcPitch,
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 srcWidth,
    N_int32 srcHeight,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 dstWidth,
    N_int32 dstHeight,
    N_int32 doClip,
    N_int32 clipLeft,
    N_int32 clipTop,
    N_int32 clipRight,
    N_int32 clipBottom,
    N_int32 mix)
```

### Prototype In

snap/graphics.h

### Parameters

<i>srcAddr</i>	Address of source bitmap in system memory
<i>srcPhysAddr</i>	Physical address of source bitmap in system memory
<i>srcPitch</i>	Pitch of source bitmap in bytes
<i>srcLeft</i>	Left coordinate of the source rectangle to copy
<i>srcTop</i>	Top coordinate of the source rectangle to copy
<i>srcWidth</i>	Width of the source rectangle in pixels
<i>srcHeight</i>	Height of the source rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>dstWidth</i>	Width of the destination rectangle in pixels
<i>dstHeight</i>	Height of the destination rectangle in scanlines
<i>doClip</i>	True if the blit should be clipped, false if not
<i>clipLeft</i>	Left coordinate for clip rectangle (inclusive)
<i>clipTop</i>	Top coordinate for clip rectangle (inclusive)
<i>clipRight</i>	Right coordinate for clip rectangle (exclusive)
<i>clipBottom</i>	Bottom coordinate for clip rectangle (exclusive)
<i>mix</i>	Mix code for the copy ( <i>GA_mixCodesType</i> )

### Description

This function copies a bitmap from system memory with a physical starting address of *srcPhysAddr* to video memory with either stretching or shrinking. This routine will copy the rectangular region of video memory from (*srcLeft*, *srcTop*, *srcLeft*+*srcWidth*-1, *srcTop*+*srcHeight*-1) to (*dstLeft*, *dstTop*, *dstLeft*+*dstWidth*-1, *dstTop*+*dstHeight*-1)

within video memory. Note that the source and destination rectangle dimensions may be different in, which is the case for doing a copy with bitmap stretching or shrinking. The *srcPhysAddr* value points to the start of the bitmap data in system memory as a *physical* memory address, not a linear memory address that the application software normally deals with. It is up to the calling application to use the necessary OS services to allocate a block of contiguous physical memory for the bitmap data, and to obtain the physical memory address to be passed into this function. Note that the source and destination rectangle dimensions may be different in, which is the case for doing a copy with bitmap stretching or shrinking.

This version is different to the *StretchBltSys* function in that the bitmap data is copied using Bus Mastering by the graphics accelerator, which allows this function to return before the copy has completed and the accelerator will complete the copy in the background with a DMA Bus Master operation. If this hardware supports Bus Mastering and this function is available, it will usually be the fastest method to copy a block of system memory to video memory.

Note that the *srcLeft* and *srcTop* coordinates define an offset within the source bitmap to be copied, so it will copy only a portion of the memory bitmap.

If the *doClip* parameter is true, then the output of the stretch function will be clipped against the passed in destination clip rectangle.

**See Also**

*StretchBlt, StretchBltLin, StretchBltSys, SrcTransBlt, DstTransBlt, BitBlt*

## StretchBltLin

Copy a linear block of video memory to another location in video memory with stretching or shrinking.

### Declaration

```
void NAPI GA_2DRenderFuncs::StretchBltLin(
    N_int32 srcOfs,
    N_int32 srcPitch,
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 srcWidth,
    N_int32 srcHeight,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 dstWidth,
    N_int32 dstHeight,
    N_int32 doClip,
    N_int32 clipLeft,
    N_int32 clipTop,
    N_int32 clipRight,
    N_int32 clipBottom,
    N_int32 mix)
```

### Prototype In

snap/graphics.h

### Parameters

<i>srcOfs</i>	Offset of source bitmap in video memory
<i>srcPitch</i>	Pitch of source bitmap in bytes
<i>srcLeft</i>	Left coordinate of the source rectangle to copy
<i>srcTop</i>	Top coordinate of the source rectangle to copy
<i>srcWidth</i>	Width of the source rectangle in pixels
<i>srcHeight</i>	Height of the source rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>dstWidth</i>	Width of the destination rectangle in pixels
<i>dstHeight</i>	Height of the destination rectangle in scanlines
<i>doClip</i>	True if the blit should be clipped, false if not
<i>clipLeft</i>	Left coordinate for clip rectangle (inclusive)
<i>clipTop</i>	Top coordinate for clip rectangle (inclusive)
<i>clipRight</i>	Right coordinate for clip rectangle (exclusive)
<i>clipBottom</i>	Bottom coordinate for clip rectangle (exclusive)
<i>mix</i>	Mix code for the copy ( <i>GA_mixCodesType</i> )

### Description

This function copies a linear region of video memory from one location to another with either stretching or shrinking. Note that the value of *srcOfs* must be aligned to the boundary specified in the *BitmapStartAlign* member of the *GA\_modeInfo* structure, and the *srcPitch* value must be padded to multiples of the *BitmapStridePad* member of the *GA\_modeInfo* structure. This routine will copy the rectangular region of video memory

from (srcLeft, srcTop, srcLeft+srcWidth-1, srcTop+srcHeight-1) to (dstLeft, dstTop, dstLeft+dstWidth-1, dstTop+dstHeight-1) within video memory. Note that the source and destination rectangle dimensions may be different in, which is the case for doing a copy with bitmap stretching or shrinking. The results of this routine are undefined if the video memory regions overlap.

Note that the srcLeft and srcTop coordinates define an offset within the source bitmap to be copied, so it will copy only a portion of an offscreen bitmap. This is useful for storing multiple images in a single offscreen bitmap, or for handling the case of software clipping offscreen bitmaps if the destination lies outside of the software clip rectangle for the destination buffer.

This version is different to the standard *BitBltFx* function in that the source bitmap to be copied can be non-conforming, and can have a different logical scanline width to the destination bitmap. This allows the bitmaps to be stored contiguously in offscreen video memory, rather than requiring the offscreen video memory to be divided up into rectangular regions, resulting in more efficient use of available offscreen memory for bitmap storage.

If the doClip parameter is true, then the output of the stretch function will be clipped against the passed in destination clip rectangle.

---

**Note:** *The value of srcOfs must be aligned to the boundary specified in the BitmapStartAlign member of the GA\_modeInfo structure, and the dstPitch value must be padded to multiples of the BitmapStridePad member of the GA\_modeInfo structure.*

---

#### **See Also**

*StretchBlt, StretchBltSys, StretchBltBM, SrcTransBlt, DstTransBlt, BitBlt*

## StretchBltSys

Copy a block of system memory to another location in video memory with stretching or shrinking.

### Declaration

```
void NAPI_GA_2DRenderFuncs::StretchBltSys(
    void *srcAddr,
    N_int32 srcPitch,
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 srcWidth,
    N_int32 srcHeight,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 dstWidth,
    N_int32 dstHeight,
    N_int32 doClip,
    N_int32 clipLeft,
    N_int32 clipTop,
    N_int32 clipRight,
    N_int32 clipBottom,
    N_int32 mix,
    N_int32 flipY)
```

### Prototype In

snap/graphics.h

### Parameters

<i>srcAddr</i>	Address of source bitmap in system memory
<i>srcPitch</i>	Pitch of source bitmap in bytes
<i>srcLeft</i>	Left coordinate of the source rectangle to copy
<i>srcTop</i>	Top coordinate of the source rectangle to copy
<i>srcWidth</i>	Width of the source rectangle in pixels
<i>srcHeight</i>	Height of the source rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>dstWidth</i>	Width of the destination rectangle in pixels
<i>dstHeight</i>	Height of the destination rectangle in scanlines
<i>doClip</i>	True if the blit should be clipped, false if not
<i>clipLeft</i>	Left coordinate for clip rectangle (inclusive)
<i>clipTop</i>	Top coordinate for clip rectangle (inclusive)
<i>clipRight</i>	Right coordinate for clip rectangle (exclusive)
<i>clipBottom</i>	Bottom coordinate for clip rectangle (exclusive)
<i>mix</i>	Mix code for the copy ( <i>GA_mixCodesType</i> )
<i>flipY</i>	True if the image should be flipped vertically

### Description

This function copies a linear region of video memory from one location to another with either stretching or shrinking. This routine will copy the rectangular region of video memory from (srcLeft, srcTop, srcLeft+srcWidth-1, srcTop+srcHeight-1) to (dstLeft, dstTop, dstLeft+dstWidth-1, dstTop+dstHeight-1) within video memory. Note that the



source and destination rectangle dimensions may be different in, which is the case for doing a copy with bitmap stretching or shrinking. The results of this routine are undefined if the video memory regions overlap.

Note that the `srcLeft` and `srcTop` coordinates define an offset within the source bitmap to be copied, so it will copy only a portion of the memory bitmap.

If the `doClip` parameter is true, then the output of the stretch function will be clipped against the passed in destination clip rectangle.

**See Also**

*StretchBlt, StretchBltLin, StretchBltBM, SrcTransBlt, DstTransBlt, BitBlt*

## UpdateScreen

Update the screen from the shadow buffer for the specified rectangle

### Declaration

```
void NAPI_GA_2DRenderFuncs::UpdateScreen(
    N_int32 left,
    N_int32 top,
    N_int32 width,
    N_int32 height)
```

### Prototype In

snap/graphics.h

### Parameters

<i>left</i>	Left coordinate of screen to update
<i>top</i>	Top coordinate of screen to update
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines

### Description

This function is used to update the screen from a shadow buffer if the application or shell driver did any custom drawing to the shadow buffer without using the SNAP driver functions. If the mode is using a shadow buffer (usually only for 4bpp modes and 8bpp banked modes) this function will be implemented and must be used, otherwise the function will be NULL.

This function may also be used by rotation, flipped and other geometry changing filter drivers to ensure that any custom drawing is properly managed on the hardware framebuffer.

## GA\_2DStateFuncs

### Prototype In

snap/graphics.h

### Description

Function group containing all the device driver functions related to managing the hardware 2D graphics accelerator state. This group of functions does not contain any functions that do any drawing on the screen, just state management.

Generally applications or shell drivers should request this block of functions from the 2d reference rasteriser library, not directly from the graphics accelerator. This will allow the library to fill in all rendering functions with software rendering as necessary automatically.

---

**Note:** *Be sure to fill in the dwSize member of this structure when you call GA\_queryFunctions to the correct size of the structure at compile time!*

---

## BuildTranslateVector

Build a color palette translation vector

### Declaration

```
void NAPI_GA_2DStateFuncs::BuildTranslateVector(
    GA_color *translate,
    GA_palette *dstPal,
    GA_palette *srcPal,
    int srcColors)
```

### Prototype In

snap/graphics.h

### Parameters

<i>translate</i>	Place to store resulting translate vector
<i>dstPal</i>	Destination palette to map to
<i>srcPal</i>	Source palette to map from
<i>srcColors</i>	Number of colors in the source palette to map

### Description

This function builds a color palette translation vector and returns it. For color index modes ( $\leq 8$ bpp), the source palette values are mapped onto the destination palette values, and the resulting translation vector will be an array of 32-bit color palette index values. To find the final color for a color value, take the source color index value and dereference it via the translation table to get the resulting color index value.

For 15bpp and higher RGB display modes, the destination palette parameter is ignored and the resulting translation vector will be a 32-bit array of packed color values that represent the source palette entries. I.e: for 15bpp modes, the resulting vector will contain 32-bit values between 0 and 0x7FFF that represent the packed color values in 5:5:5 format. Hence converting paletted bitmap data to the destination pixel format is simply a matter of looking up the entry in the translation vector and using that to store the resulting value in the framebuffer.

This function is mostly intended for applications and shell drivers that know in advance that a number of bitmaps will be translated, so can create the color translation vector once and then reuse it for all bitmaps.

## DisableDirectAccess

Disables direct access to display memory.

### Declaration

```
void NAPI_GA_2DStateFuncs::DisableDirectAccess(void)
```

### Prototype In

snap/graphics.h

### Description

This function disables direct framebuffer access and turns on the hardware accelerator again. The primary purpose of this function is to correctly arbitrate video memory access between the accelerator and the application. You *must* call this function before you perform any accelerated rendering again if the *EnableDirectAccess* function pointer was not NULL.

Note that if you are using the buffer manager functions, you should avoid using this function but instead use the *LockBuffer* and *UnlockBuffer* functions to hide the complexity of managing offscreen memory blocks.

### See Also

*EnableDirectAccess*, *WaitTillIdle*, *UnlockBuffer*

## EnableDirectAccess

Enables direct access to framebuffer memory.

### Declaration

```
void NAPI_GA_2DStateFuncs::EnableDirectAccess(void)
```

### Prototype In

snap/graphics.h

### Description

This function disables the accelerator and turns on direct framebuffer access. The primary purpose of this function is to correctly arbitrate video memory access between the accelerator and the CPU. You *must* call this function before you perform any direct rendering to the video memory if the function pointer for this function is not NULL. If the function pointer is NULL, then the controller does not need to arbitrate access and this function is not necessary (instead simply call *WaitTillIdle* before accessing the framebuffer memory).

Note that if you are using the buffer manager functions, you should avoid using this function but instead use the *LockBuffer* and *UnlockBuffer* functions to hide the complexity of managing offscreen memory blocks.

---

**Note:** *This function does an implicit WaitTillIdle when it is called to ensure that the graphics accelerator has finished all current rendering operations before enabling direct access to display memory.*

---

### See Also

*DisableDirectAccess, WaitTillIdle, LockBuffer*

## IsIdle

Determines if the graphics accelerator is idle.

### Declaration

```
N_int32 NAPI GA_2DStateFuncs::IsIdle(void)
```

### Prototype In

snap/graphics.h

### Return Value

not 0 if the accelerator is idle, 0 if not.

### Description

This function is the similar to the *WaitTillIdle* function, however it does not wait for the engine to become idle but instead returns immediately with the current status.

### See Also

*WaitTillIdle*, *EnableDirectAccess*, *DisableDirectAccess*

## Set8x8ColorPattern

Download an 8x8 color pattern to the driver.

### Declaration

```
void NAPI_GA_2DStateFuncs::Set8x8ColorPattern(
    N_int32 index,
    GA_colorPattern *pattern)
```

### Prototype In

snap/graphics.h

### Parameters

<i>index</i>	Index of the pattern to download
<i>pattern</i>	Pointer to pattern data to download

### Description

This downloads one of 8, 8x8 color patterns for all subsequent color pattern filled functions. The 8x8 color fill pattern is used for rectangle and scanline filling, where the pattern is X and Y coordinate aligned with the left edge and top edge of the display. Thus the colors in the pattern that applies to a specific pixel in the scanline is determined by the pixel's X starting at the left. Hence pixel 0 corresponds to color 0, pixel 1 = color 1 etc. It is the responsibility of the calling application to rotate the pattern before calling this routine if it is desired that the pattern be aligned to a different starting coordinate (such as with Windows Bitmaps and setting the bitmap origin). The color pattern is represented as an 8x8 array of packed pixel data. In 8bpp modes there is 8 bytes per line, for 16bpp modes there are 16bytes per line, for 24bpp modes there are 24bytes per line and for 32bpp modes there are 32 bytes per line. Hence the size of the pattern data is different depending on the color depth currently active. Each pixel color value is packed for the appropriate display mode.

---

**Note:** 8 cached patterns are supported because some hardware supports caching multiple patterns in offscreen video memory for maximum performance. In cases where the hardware only supports a single hardware pattern, the driver is responsible for caching the pattern data internally and downloading it as efficiently as possible to the display hardware.

---

### See Also

Set8x8MonoPattern, Use8x8MonoPattern, Use8x8ColorPattern,  
GA\_2DRenderFuncs::DrawColorPattScan, DrawColorPattScanList, DrawColorPattRect,  
DrawColorPattTrap



## Set8x8MonoPattern

Download an 8x8 monochrome pattern to the driver.

### Declaration

```
void NAPI_GA_2DStateFuncs::Set8x8MonoPattern(
    N_int32 index,
    GA_pattern *pattern)
```

### Prototype In

snap/graphics.h

### Parameters

<i>index</i>	Index of the pattern to download
<i>pattern</i>	Pointer to pattern data to download

### Description

This downloads one of 8, 8x8 monochrome patterns for all subsequent monochrome pattern filled functions. The 8x8 monochrome fill pattern is used for rectangle and scanline filling, where the pattern is X and Y coordinate aligned with the left edge and top edge of the display. In the bitmap pattern, pixel 0 corresponds to bit 7 in byte 0, pixel 1 = bit 6 in byte 0, ... pixel 8 = bit 7 in byte 1 etc. It is the responsibility of the calling application to rotate the pattern before calling this routine if it is desired that the pattern be aligned to a different starting coordinate (such as with Windows Bitmaps and setting the bitmap origin). The bitmap pattern is passed as a packed array of 8 bytes.

---

**Note:** 8 cached patterns are supported because some hardware supports caching multiple patterns in offscreen video memory for maximum performance. In cases where the hardware only supports a single hardware pattern, the driver is responsible for caching the pattern data internally and downloading it as efficiently as possible to the display hardware.

---

### See Also

*Use8x8MonoPattern, Set8x8ColorPattern, Use8x8ColorPattern, GA\_2DRenderFuncs::DrawPattScan, DrawPattScanList, DrawPattRect, DrawPattTrap*

## SetAlphaValue

Set the constant alpha value for blending operations.

### Declaration

```
void NAPI_GA_2DStateFuncs::SetAlphaValue(  
    N_uint8 alpha)
```

### Prototype In

snap/graphics.h

### Parameters

*alpha*                      New constant alpha value to make active

### Description

This function sets the constant alpha value for subsequent drawing operations. The constant alpha value is used when either the source or destination blending functions include constant alpha, otherwise this value is ignored.

---

**Note:** *Not all hardware supports this function, and if the hardware does not support it this function will be NULL.*

---

### See Also

*SetBlendFunc*

## SetBackColor

Set the background color for subsequent rendering operations.

### Declaration

```
void NAPI_GA_2DStateFuncs::SetBackColor(  
    GA_color color)
```

### Prototype In

snap/graphics.h

### Parameters

*color*                      New background color to set

### Description

This function sets the hardware color for subsequent accelerated rendering primitives. For solid primitives such as DrawRect only the foreground color is used. For patterns primitives such as DrawPattRect and DrawStippleLine both the foreground and background colors are used.

### See Also

*SetForeColor, SetMix, Set8x8MonoPattern, Set8x8ColorPattern*

## SetBlendFunc

Set the source and destination blending functions for subsequent drawing operations.

### Declaration

```
void NAPI GA_2DStateFuncs::SetBlendFunc(
    N_int32 srcBlendFunc,
    N_int32 dstBlendFunc)
```

### Prototype In

snap/graphics.h

### Parameters

<i>srcBlendFunc</i>	New source blending function ( <i>GA_blendFuncType</i> )
<i>dstBlendFunc</i>	New destination blending function ( <i>GA_blendFuncType</i> )

### Description

This function sets the source and destination blending function for subsequent drawing operations. The supported source and destination blending operations are defined in the *GA\_blendFuncType* enumeration, and the final result is a combination of the source and destination blending functions. Blending is disabled by calling this function with *srcBlendFunc* or *dstBlendFunc* set to *gaBlendNone*. Before any blending will take effect, both the source and the destination blending functions must be enabled.

---

**Note:** *Not all hardware supports this function, and if the hardware does not support it this function will be NULL.*

---

### See Also

*SetAlphaValue*

## SetDrawBuffer

Sets a display buffer as the active drawing buffer.

### Declaration

```
N_int32 NAPI GA_2DStateFuncs::SetDrawBuffer(
    GA_buffer *drawBuf)
```

### Prototype In

snap/graphics.h

### Parameters

*drawBuf*                      Buffer to make the active drawing buffer.

### Return Value

0 on success, -1 on failure

### Description

This function allows the application to make a display memory buffer the active rendering buffer for all subsequent drawing commands. The display memory drawing buffer may be a region of memory with non-conforming dimensions compared to the main display mode (ie: a 320x240 offscreen buffer with an 800x600 display mode). However if the hardware cannot support non-conforming regions, this function will fail. It may also fail if you request a drawing buffer where the offset is not aligned on a scanline boundary for hardware that does not support rendering to arbitrary offscreen buffers.

Note also that some hardware has restrictions on the alignment of both the starting offset in display memory and the pitch in display memory. The `BitmapStartAlign` and `BitmapStridePad` fields of the *GA\_modeInfo* structure indicate the alignment requirements, so you *must* ensure that the `Offset` and `Stride` members of the *GA\_buffer* structure are correctly aligned based on these values (if not this function will fail). Some hardware may also require the use of the *AlignLinearBuffer* command as well.

In order to avoid the complexity of managing offscreen memory, application programmers and shell driver programmers should use the buffer manager functions instead (*GA\_bufferFuncs*). These functions provide a more abstract interface to offscreen video memory, and will automatically take care of managing all the details of surface allocation for you.

---

**Note:** *This function should never fail if the starting address is aligned to a scanline boundary and the scanline width is the same as the logical display pitch for the display mode.*

---

### See Also

*GA\_bufferFuncs*, *SetDrawBuffer*

## SetForeColor

Set the foreground color for subsequent rendering operations.

### Declaration

```
void NAPI_GA_2DStateFuncs::SetForeColor(  
    GA_color color)
```

### Prototype In

snap/graphics.h

### Parameters

*color*                      New foreground color to set

### Description

This function sets the hardware foreground color for subsequent accelerated rendering primitives. For solid primitives such as DrawRect only the foreground color is used. For patterns primitives such as DrawPattRect and DrawStippleLine both the foreground and background colors are used.

### See Also

*SetBackColor, SetMix, Set8x8MonoPattern, Set8x8ColorPattern*

## SetLineStipple

Sets the current 16-bit line stipple pattern.

### Declaration

```
void NAPI_GA_2DStateFuncs::SetLineStipple(  
    GA_stipple stipple)
```

### Prototype In

snap/graphics.h

### Parameters

*stipple*                      New 16-bit line stipple pattern to set

### Description

This function sets up a 16-bit line stipple for all subsequent stippled line drawing functions. In the stipple, pixel 0 corresponds to bit 0, pixel 1 = bit 1, ... pixel 15 = bit 15 etc. If the stippled line is drawn in transparent mode, where a bit is 0 in the stipple pattern the destination pixel remains untouched. If the stipple line is drawn in opaque mode, where a bit is 0 in the stipple pattern the destination pixel is drawn in the background color. In all cases where a bit in the stipple pattern is 1, the pixel is drawn in the foreground color.

---

**Note:** *When a new stipple pattern is downloaded, the line stipple count is reset back to 0.*

---

### See Also

*SetLineStippleCount, SetLineStyle, DrawStippleLineInt, DrawBresenhamStippleLine*

## SetLineStippleCount

Set the 32-bit line stipple count

### Declaration

```
void NAPI_GA_2DStateFuncs::SetLineStippleCount(  
    N_uint32 count)
```

### Prototype In

snap/graphics.h

### Parameters

*count*                      New 32-bit line stipple count

### Description

This function sets the line stipple count for stippled line drawing. When a line is drawn, every pixel that is drawn in the line increments the stipple counter by 1. The stipple counter modules 16 is used to determine which bit in the stipple pattern should be used for the next pixel drawn in the line, and because the driver maintains this stipple count across stippled line drawing functions, it allows a single stipple pattern to be correctly applied to a number of connected line segments. This function allows the user application to preset the stipple count to a specified value before drawing the next stippled line.

### See Also

*SetLineStipple*, *SetLineStyle*, *DrawStippleLineInt*, *DrawBresenhamStippleLine*



## SetLineStyle

Set the OS/2 style line style parameters for drawing patterned lines

### Declaration

```
void NAPI_GA_2DStateFuncs::SetLineStyle(
    N_uint32 styleMask,
    N_uint32 styleStep,
    N_uint32 styleValue)
```

### Prototype In

snap/graphics.h

### Parameters

<i>styleMask</i>	32-bit style mask for styled lines
<i>styleStep</i>	8-bit value added to styleValue for each major pixel
<i>styleValue</i>	The style value for the first pixel in the line

### Description

This function sets up the parameters for drawing 32-bit OS/2 style styled lines for all subsequent styled line drawing functions. In the styleMask, pixel 0 corresponds to bit 0, pixel 1 = bit 1, ... pixel 31 = bit 31 etc. If the styled line is drawn in transparent mode, where a bit is 0 in the styleMask the destination pixel remains untouched. If the styled line is drawn in opaque mode, where a bit is 0 in the styleMask the destination pixel is drawn in the background color. In all cases where a bit in the styleMask pattern is 1, the pixel is drawn in the foreground color.

The styleValue passed in is composed of an error value and a mask position as follows:

```
|=====|
|  high word  |  3 bits  |  5 bits  |  8 bits  |
|=====|
|  not used   | not used | mask pos | error value |
|=====|
```

The error value determines the error value at the first pixel in the line. The mask position is an index into the styleMask to determine how the pixel should be drawn as outlined above.

### See Also

*SetLineStipple, SetLineStippleCount, DrawStyleLineInt, DrawBresenhamStyleLine*

## SetMix

Set the mix code for subsequent rendering operations.

### Declaration

```
N_int32 NAPI GA_2DStateFuncs::SetMix(
    N_int32 mix)
```

### Prototype In

snap/graphics.h

### Parameters

*mix*                      New mix code to set (*GA\_mixCodesType*)

### Return Value

True on success, false if mix is not supported.

### Description

This function sets the hardware mix operation for subsequent accelerated rendering primitives. The mix does not change that often, and is usually only set once for a range of rendering primitives, so it is set here via a state function rather than being passed to each rendering functions (unlike the colors which change constantly). The mix modes are defined in the *GA\_mixCodesType* enumeration.

---

**Note:** *If the hardware does not support a particular mix this function may return false. It is up to the calling code to detect this and properly fall back on software rendering to handle this particular mix (the 2d reference rasteriser does this automatically).*

---

### See Also

*SetForeColor, SetBackColor, Set8x8MonoPattern, Set8x8ColorPattern*

## SetPlaneMask

Set the hardware plane mask for subsequent drawing operations.

### Declaration

```
void NAPI_GA_2DStateFuncs::SetPlaneMask(  
    N_uint32 mask)
```

### Prototype In

snap/graphics.h

### Parameters

*mask*                      New plane mask to make active

### Description

This function sets the hardware plane mask for the hardware, which is used to mask out specific bits from being affected during writes to the framebuffer. The mask passed in should be a packed color value for the currently active display mode.

---

**Note:** *Not all hardware supports this function, and if the hardware does not support it this function will be NULL.*

---

## Use8x8ColorPattern

Selects one of the 8x8 color patterns for subsequent rendering operations.

### Declaration

```
void NAPI_GA_2DStateFuncs::Use8x8ColorPattern(  
    N_int32 index)
```

### Prototype In

snap/graphics.h

### Parameters

*index*                      Index of the 8x8 color pattern to make active

### Description

This function selects one of the 8, 8x8 color patterns to be used for all subsequent color pattern filled functions.

### See Also

*Set8x8MonoPattern, Use8x8MonoPattern, Set8x8ColorPattern, GA\_2DRenderFuncs::DrawColorPattScan, DrawColorPattScanList, DrawColorPattRect, DrawColorPattTrap*

## Use8x8MonoPattern

Selects one of the 8x8 monochrome patterns for subsequent rendering operations.

### Declaration

```
void NAPI_GA_2DStateFuncs::Use8x8MonoPattern(  
    N_int32 index)
```

### Prototype In

snap/graphics.h

### Parameters

*index*                      Index of the 8x8 mono pattern to make active

### Description

This function selects one of the 8, 8x8 monochrome patterns to be used for all subsequent monochrome pattern filled functions. This function enables the monochrome pattern to be used for opaque mono pattern fills. This means that where a bit is 0 in the bitmap pattern the destination pixel is drawn in the background color, and where a bit is 1 in the bitmap pattern the destination pixel is drawn in the foreground color.

### See Also

*Use8x8TransMonoPattern, Set8x8MonoPattern, Set8x8ColorPattern, Use8x8ColorPattern, GA\_2DRenderFuncs::DrawPattScan, DrawPattScanList, DrawPattRect, DrawPattTrap*

## Use8x8TransColorPattern

Selects one of the 8x8 color patterns for subsequent rendering operations.

### Declaration

```
void NAPI_GA_2DStateFuncs::Use8x8TransColorPattern(
    N_int32 index,
    GA_color transparent)
```

### Prototype In

snap/graphics.h

### Parameters

<i>index</i>	Index of the 8x8 color pattern to make active
<i>transparent</i>	Transparent color for the pattern fill

### Description

This function selects one of the 8, 8x8 color patterns to be used for all subsequent color pattern filled functions. This version is similar to the regular *Use8x8ColorPattern* function, however it enables transparency for the color pattern. Where a pixel in the pattern is equal to the transparent color, the destination pixel will remain untouched, otherwise the destination pixel will take on the color of the pixel in the bitmap pattern.

---

**Note:** *Some hardware may not support transparent color pattern fills, in which case this function will be a NULL pointer.*

---

### See Also

*Set8x8MonoPattern, Use8x8MonoPattern, Set8x8ColorPattern, GA\_2DRenderFuncs::DrawColorPattScan, DrawColorPattScanList, DrawColorPattRect, DrawColorPattTrap*

## Use8x8TransMonoPattern

Selects one of the 8x8 monochrome patterns for subsequent rendering operations.

### Declaration

```
void NAPI_GA_2DStateFuncs::Use8x8TransMonoPattern(
    N_int32 index)
```

### Prototype In

snap/graphics.h

### Parameters

*index*                      Index of the 8x8 mono pattern to make active

### Description

This function selects one of the 8, 8x8 monochrome patterns to be used for all subsequent monochrome pattern filled functions. This function enables the monochrome pattern to be used for transparent mono pattern fills. This means that where a bit is 0 in the bitmap pattern the destination pixel remains untouched, and where a bit is 1 in the bitmap pattern the destination pixel is drawn in the foreground color.

---

**Note:** *Some hardware may not support transparent mono pattern fills, in which case this function will be a NULL pointer.*

---

### See Also

*Use8x8MonoPattern, Set8x8MonoPattern, Set8x8ColorPattern, Use8x8ColorPattern, GA\_2DRenderFuncs::DrawPattScan, DrawPattScanList, DrawPattRect, DrawPattTrap*

## WaitTillIdle

Waits until the graphics accelerator is idle.

### Declaration

```
void NAPI_GA_2DStateFuncs::WaitTillIdle(void)
```

### Prototype In

snap/graphics.h

### Description

This function waits until the hardware accelerator has completed all currently queued rendering operations. The primary purpose of this function is to provide the application with the ability to ensure all rendering is complete, before swapping display pages when doing double buffering, or before directly accessing the framebuffer memory.

---

**Note:** *This function is required to wait for the entire graphics engine to be idle, including waiting until any pending DMA or bus master operations have completed.*

---

### See Also

*EnableDirectAccess, DisableDirectAccess, IsIdle*



## GA\_AccelFlagsType

### Declaration

```
typedef enum {
    gaAccelType_Custom    = 0,
    gaAccelType_Full      = 1,
    gaAccelType_Most      = 2,
    gaAccelType_Basic     = 3,
    gaAccelType_None      = 4,
} GA_AccelFlagsType
```

### Prototype In

snap/graphics.h

### Description

Definitions for values stored in the `accelType` member of the `GA_Options` structure. These flags define the different values for this particular option which is multi-state and not just an on/off option. This option is designed to be used as a fallback measure in the field to disable problem functions in a driver that an end user might be experiencing.

The `gaAccelType_Custom` value indicates that the user has manually overridden specific hardware acceleration options rather than using the four pre-defined settings.

The `gaAccelType_Full` value indicates that full hardware acceleration should be used, and is always the default option for all drivers.

The `gaAccelType_Most` value indicates that most acceleration functions should be used in the driver. This basically disables support for hardware mouse cursor and hardware video overlays functionality.

The `gaAccelType_Basic` value indicates that only basic acceleration functions should be used in the driver. Basic acceleration includes solid fills, mono and color pattern fills and screen to screen blits. All other functions are disabled.

`gaAccelType_None` value indicates that no hardware acceleration functions should be used in the driver.

### Members

<i>gaAccelType_Custom</i>	Custom hardware acceleration
<i>gaAccelType_Full</i>	Full hardware acceleration (default)
<i>gaAccelType_Most</i>	Most hardware acceleration
<i>gaAccelType_Basic</i>	Basic hardware acceleration
<i>gaAccelType_None</i>	No hardware acceleration

## GA\_AttributeExtFlagsType

### Declaration

```
typedef enum {
    gaIsPanningMode           = 0x00000001,
    gaNoRefreshCtrl           = 0x00000002
} GA_AttributeExtFlagsType
```

### Prototype In

snap/graphics.h

### Description

Flags for the AttributesExt member of the *GA\_modeInfo* structure and in the AttributesExt member of the main *GA\_devCtx* device context block structure. These flags define the hardware capabilities of the particular device or graphics mode.

The gaIsPannedMode flag is used to determine if the mode is a virtual hardware panned display mode or if the mode is a non-panned display mode. This flag is only ever set if you call the GetVideoModeInfoExt or GetCustomVideoModeInfoExt functions, as the original versions of these functions assume non panning modes will be reported. If the mode is a hardware panned mode for the requested output device, it means that if the mode is set while that output device is active hardware panning will be enabled. It is then up to the shell driver to interface with the mouse driver to implement the actual hardware panning.

The gaNoRefreshCtrl flag indicates that the device has no refresh control. This will only be reported in the *GA\_devCtx* variable, and is only so that the VBE/Core fallback driver can indicate whether refresh control is available or not.

### Members

<i>gaIsPanningMode</i>	Mode is a virtual hardware panning display mode
<i>gaNoRefreshCtrl</i>	Device has no refresh rate control

## GA\_AttributeFlagsType

### Declaration

```
typedef enum {
    gaHaveDisplayStart          = 0x00000001,
    gaHaveBankedBuffer          = 0x00000002,
    gaHaveLinearBuffer          = 0x00000004,
    gaHaveAccel2D                = 0x00000008,
    gaHaveHWCursor              = 0x00000010,
    gaHave8BitDAC                = 0x00000020,
    gaHaveNonVGAMode            = 0x00000040,
    gaHaveDoubleScan            = 0x00000080,
    gaHaveTripleScan            = 0x00000100,
    gaHaveInterlaced            = 0x00000200,
    gaHaveTripleBuffer          = 0x00000400,
    gaHaveStereo                 = 0x00000800,
    gaHaveHWStereoSync          = 0x00001000,
    gaHaveEVCStereoSync         = 0x00002000,
    gaHaveAccelVideo            = 0x00004000,
    gaHaveAccel3D               = 0x00008000,
    gaHave8bppRGBCursor         = 0x00010000,
    gaHaveAccelIOPL             = 0x00040000,
    gaHaveEngineClock           = 0x00200000,
    gaIsGUIDesktop              = 0x01000000,
    gaIsVirtualMode             = 0x08000000,
    gaHaveMultiHead             = 0x02000000,
    gaHaveDFPOutput             = 0x04000000,
    gaHaveLCDOutput             = 0x10000000,
    gaHaveTVOutput              = 0x20000000,
    gaIsTextMode                = 0x40000000
} GA_AttributeFlagsType
```

### Prototype In

snap/graphics.h

### Description

Flags for the Attributes member of the *GA\_modelInfo* structure and in the Attributes member of the main *GA\_devCtx* device context block structure. These flags define the hardware capabilities of the particular device or graphics mode.

The *gaHaveDisplayStart* flag is used to determine whether the graphics mode supports changing the CRTC display start address. This is used to implement hardware virtual scrolling and multi-buffering for flicker free animation. If this bit is 0, then the application cannot change the display start address after initialising a display mode.

The *gaHaveBankedBuffer* flag is used to determine if the graphics mode supports the banked framebuffer access modes. If this bit is 0, then the application cannot use the banked framebuffer style access. Some controllers may not support a banked framebuffer mode in some modes. In this case a linear framebuffer mode will be provided (either a banked buffer or linear buffer must be available for the mode to be valid).

The `gaHaveLinearBuffer` flag is used to determine if the graphics mode supports the linear framebuffer access modes. If this bit is 0, then the application cannot start the linear framebuffer graphics mode.

The `gaHaveAccel2D` flag is used to determine if the graphics mode supports 2D accelerator functions. If this bit is 0, then the application can only use direct framebuffer access in this video mode, and the 2D acceleration functions are not available. The cases where this might crop up are more common than you might think. This bit may be 0 for very low resolution graphics modes on some controllers, and on older controllers for the 24 bit and above graphics modes.

The `gaHaveHWCursor` flag is used to determine if the controller supports a hardware cursor for the specified graphics mode. You must check this flag for each graphics mode before attempting to use the hardware cursor functions as some graphics modes will not be able to support the hardware cursor (but may still support 2D acceleration).

The `gaHave8BitDAC` flag is used to determine if the controller will be using the 8 bit wide palette DAC modes when running in 256 color index modes. The 8 bit DAC modes allow the palette to be selected from a range of 16.7 million colors rather than the usual 256k colors available in 6 bit DAC mode. The 8 bit DAC mode allows the 256 color modes to display a full range of 256 grayscales, while the 6 bit mode only allows a selection of 64 grayscales. Note that the 8 bit DAC mode is not selectable. If the hardware supports an 8 bit DAC, it will always be used by default.

The `gaHaveNonVGAMode` flag is used to determine if the mode is a VGA compatible mode or a NonVGA mode. If this flag is set, the application software must ensure that no attempts are made to directly program any of the standard VGA compatible registers such as the RAMDAC control registers and input status registers while the NonVGA graphics mode is used. Attempting to use these registers in NonVGA modes generally results in the application program hanging the system.

The `gaHaveDoubleScan` flag is used to determine if the mode requires double scanning. If this bit is set, the double scan bit must be set for the graphics mode if it is initialised with generic refresh control turned on.

The `gaHaveTripleScan` flag is used to determine if the mode requires triple scanning. If this bit is set, the triple scan bit must be set for the graphics mode if it is initialised with generic refresh control turned on.

The `gaHaveInterlaced` flag is used to determine if the mode supports interlaced operation or not. If this bit is set, the mode may be initialized for interlaced operation when using the refresh rate control to initialise the mode.

The `gaHaveTripleBuffer` flag is used to determine if the mode supports hardware triple buffering.

The `gaHaveStereo` flag is used to determine if the mode supports hardware support for stereo LC shutter glasses.

The `gaHaveHWStereoSync` flag is used to determine if the controller supports the hardware stereo LC shutter glasses sync signal via the VESA EVC Enhanced Video Connector. The `gaHaveEVCStereoSync` flag is used to determine if the controller supports the hardware stereo LC shutter glasses sync signal via the VESA mini-DIN3 stereo connector. If either of these values are set, the application can disable all software stereo sync mechanisms and rely on the the hardware stereo sync for maximum performance.

The `gaHaveAccelVideo` flag is used to determine if the mode supports hardware video acceleration. If this bit is not 0, then the application can use the hardware video functions for video overlay windows.

The `gaHaveAccel3D` flag is used to determine if the mode supports hardware 3D acceleration. If this bit is not 0, then the application can use the hardware 3D acceleration functions for high performance 3D graphics.

The `gaHave8bppRGBCursor` flag is used to determine if the color values for the hardware cursor in 8bpp modes are defined as a color index or as a TrueColor RGB tuple. Most cards require a color index in 8bpp modes, but some new hardware uses a TrueColor cursor in 8bpp display modes and this flag will be set if this is the case.

The `gaHaveAccelIOPL` flag indicates that the accelerated drawing functions require IOPL access to be enabled. If this flag is not set, then the 2D and 3D drawing functions use only memory mapped registers and hence can be executed entirely in ring-3 without needing IOPL to be enabled. Note that this does not include hardware cursor functions or hardware video overlay functions, only 2D and 3D drawing functions. It is assumed that all initialisation and driver functions require IOPL to be enabled.

The `gaIsVirtualMode` flag indicates that the mode is a special multi-controller virtual display mode that spans multiple display devices. This is an informational flag so that any high level OS drivers can know when one of these modes is in use.

The `gaHaveMultiHead` flag is used to determine if the controller is capable of supporting dual head operation via two separate CRTC connector output. This flag is generally only included the `GA_devCtx` Attribute member and not in the Attributes member of the `GA_modeInfo` structure.

The `gaHaveDFPOutput` flag is used to determine if a mode can be displayed on an LCD flat panel monitor using the DFP or DVI connectors. This flag is generally only available for graphics cards that have DVI or DFP connector and indicates display modes can support output to the LCD flat panel monitor as well as simultaneous output to both displays at the same time.

The `gaHaveLCDOutput` flag is used to determine if a mode can be displayed on an LCD flat panel. This flag is generally only available for laptop chipsets, and indicates display modes can support output to the LCD panel as well as simultaneous output to both displays at the same time.

The `gaHaveTVOutput` flag is used to determine if a mode can be displayed via the TVOut connector for the graphics card. If the graphics card does not support TVOut capabilities this flag will never be set. Otherwise it will be set for those display modes that can be displayed on the TV. Note that both PAL and NTSC output may be supported, or only one or the other depending on the underlying hardware.

The `gaIsTextMode` flag is used to determine if the mode is a graphics mode or a text mode. If this flag is set to 1, then the mode is a hardware text mode and not a graphics mode.

## Members

<i>gaHaveDisplayStart</i>	Mode supports changing the display start address
<i>gaHaveBankedBuffer</i>	Mode supports banked framebuffer access
<i>gaHaveLinearBuffer</i>	Mode supports linear framebuffer access
<i>gaHaveAccel2D</i>	Mode supports 2D acceleration
<i>gaHaveHWCursor</i>	Mode supports a hardware cursor
<i>gaHave8BitDAC</i>	Mode uses an 8 bit palette DAC
<i>gaHaveNonVGA Mode</i>	Mode is a NonVGA mode
<i>gaHaveDoubleScan</i>	Mode is double scanned
<i>gaHaveTripleScan</i>	Mode is triple scanned
<i>gaHaveInterlaced</i>	Mode supports interlacing
<i>gaHaveTripleBuffer</i>	Mode supports triple buffering
<i>gaHaveStereo</i>	Mode supports stereo LCD glasses
<i>gaHaveHWStereoSync</i>	Mode supports stereo signalling
<i>gaHaveEVCStereoSync</i>	Mode supports stereo sync via EVC connector
<i>gaHaveAccelVideo</i>	Mode supports video playback acceleration
<i>gaHaveAccel3D</i>	Mode supports 3D acceleration
<i>gaHave8bppRGBCursor</i>	Mode requires RGB colors for 8bpp hardware cursor
<i>gaHaveAccelIOPL</i>	Mode needs IOPL for drawing functions
<i>gaHaveEngineClock</i>	Display adapter supports programmable engine clock
<i>gaIsGUIDesktop</i>	The mode is the original GUI desktop mode
<i>gaIsVirtualMode</i>	Mode is a multi-head or multi-controller virtual mode
<i>gaHaveMultiHead</i>	Display adapter supports multi head operation
<i>gaHaveDFPOutput</i>	Mode supports output to DFP digital flat panel
<i>gaHaveLCDOutput</i>	Mode supports output to LCD laptop display
<i>gaHaveTVOutput</i>	Mode supports output to TV connector
<i>gaIsTextMode</i>	Mode is a text mode rather than a graphics mode

## GA\_BitBltFxFlagsType

### Declaration

```
typedef enum {
    gaBltMixEnable           = 0x00000001,
    gaBltStretchNearest     = 0x00000002,
    gaBltStretchXInterp     = 0x00000004,
    gaBltStretchYInterp     = 0x00000008,
    gaBltColorKeySrcSingle   = 0x00000010,
    gaBltColorKeySrcRange    = 0x00000020,
    gaBltColorKeyDstSingle   = 0x00000040,
    gaBltColorKeyDstRange    = 0x00000080,
    gaBltFlipX              = 0x00000100,
    gaBltFlipY              = 0x00000200,
    gaBltBlend               = 0x00000400,
    gaBltConvert             = 0x00000800,
    gaBltClip                = 0x00001000,
    gaBltDither              = 0x00002000,
    gaBltTranslateVec        = 0x00004000
} GA_BitBltFxFlagsType
```

### Prototype In

snap/graphics.h

### Description

Flags for hardware blitting with special effects, passed to the BltBltFx family of functions. This family of functions exposes a wide variety of special effects blitting if the hardware is capable of these functions. You can determine what special effects are supported by the hardware by examining the BitBltCaps member of the *GA\_modeInfo* structure. However to check whether a set of combined effects are supported, set the desired effects flags in the *GA\_bltFx* structure and call the BitBltFxTest function. The driver will examine the passed in flags and return true if the combination is supported, and false if not. Calling a BltBltFx function with a combination of flags not supported by the hardware will produce undefined results.

The gaBltMixEnable flag determines if the graphics mode supports arbitrary mix modes for extended BitBlt functions.

The gaBltStretchNearest flag determines if the graphics mode supports hardware stretch blitting, with nearest pixel filtering.

The gaBltStretchXInterp flag determines if the graphics mode supports hardware stretch blitting, with linear interpolated filtering in the X direction.

The gaBltStretchYInterp flag determines if the graphics mode supports hardware stretch blitting, with linear interpolated filtering in the Y direction.

The gaBltColorKeySrcSingle flag determines whether the graphics mode supports hardware source transparent blitting with single source color key. When hardware source color keying is enabled, any pixel data in the incoming bitmap that matches the currently set color key will be ignored and not displayed on the screen, essentially making those source pixels transparent.

The `gaBltColorKeySrcRange` flag determines whether the graphics mode supports hardware source transparent blitting with a range of color keys. This is the same as single source color keying, but the color key values may be allowed to fall within a range of available colors (useful if data has been filtered causing the colors to shift slightly).

The `gaBltColorKeyDstSingle` flag determines whether the graphics mode supports hardware destination transparent blitting with single destination color key. When hardware destination color keying is enabled (sometimes called blue-screening), any destination pixels in the framebuffer that match the currently set color key, will cause the source input pixels to be ignored.

The `gaBltColorKeyDstRange` flag determines whether the graphics mode supports hardware destination transparent blitting with a range of color keys. This is the same as single destination color keying, but the color key values may be allowed to fall within a range of available colors (useful if data has been filtered causing the colors to shift slightly).

The `gaBltFlipX` flag determines whether the graphics mode supports hardware blitting with data flipped in the X axis. This is useful for 2D sprite based games and animation where the same sprite data can be reused for characters going left or right on the screen by flipping the data during the blit operation.

The `gaBltFlipY` flag determines whether the graphics mode supports hardware blitting with data flipped in the Y axis. This is useful for 2D sprite based games and animation where the same sprite data can be reused for characters going up or down on the screen by flipping the data during the blit operation.

The `gaBltBlend` flag determines whether the hardware can support alpha blended blit operations.

The `gaBltConvert` flag determines whether the hardware can support pixel format conversion.

The `gaBltClip` flag determines whether the hardware can support clipping while blitting is in effect. This is usually only used to implement proper clipping for stretching operations, where software clipping can get complicated.

The `gaBltDither` flag determines whether the closest color is selected, or if dithering is used when blitting an RGB bitmap where the destination is an 8bpp, 15bpp or 16bpp device context. Dithering slows things down somewhat for 15/16bpp modes, but produces better quality. Dithering in 8bpp looks best if a halftone palette is used, and in fact is a lot faster than using the closest color method. Dithering in 8bpp will however map to any palette, but the quality is best if a halftone palette is used.

The `gaBltTranslateVec` is used to indicate that the color translation vector supplied in the `TranslateVec` member. This is used in situations where the calling code has already computed a color translation vector for the source and destination bitmaps, and will then avoid the overhead of computing the translation vector dynamically for each blit



operation. This is only useful when color converting between 4bpp and 8bpp bitmaps where the destination is also 4bpp or 8bpp (potentially with a different palette).

---

**Note:** *These flags are also passed to the BitBltFx family of functions to define the type of extended Blt that should be performed, as well as reporting the available capabilities via the GA\_bltFx structure stored in the GA\_modeInfo structure.*

**Note:** *Availability of some features may be mutually exclusive on other features. Hence you must call BitBltFxTest first to find out if the set of features that you require are all supported at the same time before attempting to perform an extended BitBlt operation.*

**Note:** *In many cases stretching with X filtering is relatively cheap, while Y filtering is more expensive. Hence it may be faster on some hardware to enable only X filtering and not Y filtering to get improved performance.*

---

## Members

<i>gaBltMixEnable</i>	Mix code enabled, defined in <i>GA_bltFx</i> structure
<i>gaBltStretchNearest</i>	Enable stretching, nearest pixel
<i>gaBltStretchXInterp</i>	Enable X axis filtering for stretch blit
<i>gaBltStretchYInterp</i>	Enable Y axis filtering for stretch blit
<i>gaBltColorKeySrcSingle</i>	Source color keying enabled, single color
<i>gaBltColorKeySrcRange</i>	Source color keying enabled, range of colors
<i>gaBltColorKeyDstSingle</i>	Destination color keying enabled, single color
<i>gaBltColorKeyDstRange</i>	Destination color keying enabled, range of colors
<i>gaBltFlipX</i>	Enable flip in X axis
<i>gaBltFlipY</i>	Enable flip in Y axis
<i>gaBltBlend</i>	Enable alpha blending
<i>gaBltConvert</i>	Enable pixel format/palette conversion
<i>gaBltClip</i>	Clip to destination clip rectangle for stretching
<i>gaBltDither</i>	Dither if an 8/15/16bpp destination
<i>gaBltTranslateVec</i>	Color translation vector supplied in TranslateVec

## GA\_BresenhamLineFlagsType

### Declaration

```
typedef enum {
    gaLineXMajor           = 0x00000001,
    gaLineXPositive        = 0x00000002,
    gaLineYPositive        = 0x00000004,
    gaLineDoLastPel        = 0x00000008,
    gaLineDoFirstPel       = 0x00000010
} GA_BresenhamLineFlagsType
```

### Prototype In

snap/graphics.h

### Description

Flags for hardware line drawing using the bresenham engine line draw function.

### Members

<i>gaLineXMajor</i>	Line is X major (ie: longer in the X direction)
<i>gaLineXPositive</i>	Direction of line is positive in X
<i>gaLineYPositive</i>	Direction of line is positive in Y
<i>gaLineDoLastPel</i>	Draw the last pixel in the line

## GA\_BufferFlagsType

### Declaration

```
typedef enum {
    gaBufferSysMem          = 0x00000001,
    gaBufferCached         = 0x00000002,
    gaBufferMoveable       = 0x00000004,
    gaBufferPageable       = 0x00000008,
    gaBufferPriority        = 0x00000010,
    gaBufferNoSysMem       = 0x00000020,
    gaBuffer3D             = 0x00000040,
    gaBufferFlippable      = 0x00010000,
    gaBufferVideo          = 0x00020000,
    gaBufferDepth          = 0x00040000,
    gaBufferTexture        = 0x00080000,
    gaBufferStencil        = 0x00100000,
    gaBufferSpecial        = 0x7FFF0000
} GA_BufferFlagsType
```

### Prototype In

snap/graphics.h

### Description

Flags for buffer flags passed to the AllocBuffer function. The flags define how the buffer is allocated, and the type of buffer.

The gaBufferSysMem flag indicates that the buffer is currently located in system memory only. It is possible for a buffer that was allocated with the gaBufferPageable and gaBufferCached flags to initially be in video memory but then get paged out to system memory to make space for higher priority buffers. You can also set this flag when you allocate a buffer to cause the buffer to be allocated in system memory instead of video memory.

The gaBufferCached flag indicates that the buffer should have a system memory cache allocated for it, so that it can be swapped in and out of video memory as necessary. Sometimes it may be useful to have buffers cached in system memory, but not have them pageable. Thus the system memory cache can be used to refresh the video memory as necessary if the video memory contents were lost (ie: on a focus switch etc). Note that the system memory cache is *not* maintained automatically by SNAP Graphics, but rather it is up to the application code to maintain the contents of the system memory cache if they need to be kept in sync. You can use the UpdateCache and UpdateFromCache functions to keep the system memory cache in sync as necessary.

The gaBufferMoveable flag indicates that the buffer should be allocated on the moveable buffer heap, so that the buffer can be moved around as necessary to compact the heap if it becomes fragmented. For buffers that should never move in video memory, this flag should not be set and the buffers will be allocated in the non-moveable or fixed heap.

The gaBufferPageable flag indicates that the buffer is a low priority buffer and can be paged to system memory in order to make room for higher priority buffers. Setting gaBufferPageable flag will automatically set the gaBufferCached flag so that there is a

system memory cache for the buffer. Pageable buffers will be paged back into video memory when the heap becomes free of all non-pageable buffers. Hence shell drivers using the buffer manager to cache bitmaps etc should make those bitmaps all pageable, so that they will get pages to system memory if applications need more offscreen memory (ie: 2D or 3D graphics intensive apps). When the graphics intensive app exits, the pageable buffers will get pages back into video memory as all non-pageable buffers will have been free.

The `gaBufferPriority` flag indicates that the buffer is a high priority buffer. As long as there are any high priority buffers still allocated, the buffer manager will not attempt to page back in pageable buffers from system memory. Hence DirectDraw application buffers etc should be marked as priority buffers, so that pageable buffers will not be brought back into video memory until the DirectDraw app exits.

The `gaBufferNoSysMem` flag is used to indicate that the surface being created should only ever be allocated in video memory. If there is no video memory available, the buffer allocation function will fail (normally it will attempt to allocate the buffer in system memory if the `gaBufferCached` or `gaBufferPageable` flags are set).

The `gaBuffer3D` flag is used to indicate that the surface being created should be capable for being the destination for 3D hardware rendering. If you need hardware 3D capabilities for the primary and flippable buffers, you should pass this flag to the `InitBuffers` function when you initialise the buffer manager.

The `gaBufferFlippable` flag is used to indicate whether the buffer is a flippable buffer that can be viewed and made visible via the `MakeVisibleBuffer` function. All flippable buffers must be the same dimensions as the primary display mode, and are allocated when you first call the `InitBuffers` function to initialise the buffer manager.

The `gaBufferVideo` flag is an internal flag used to indicate that the buffer is a video overlay window buffer.

The `gaBufferDepth` flag is an internal flag used to indicate that the buffer is a hardware depth buffer.

The `gaBufferTexture` flag is an internal flag used to indicate that the buffer is a hardware texture map.

The `gaBufferStencil` flag is an internal flag used to indicate that the buffer is a hardware stencil buffer.

---

**Note:** *These flags are also passed to the `AllocBuffer` function to determine how the buffer should be allocated. Some of the flags are internal and should never be passed to `AllocBuffer` as they are used internally. Flags above or equal to `gaVideo` are used internally to indicate what type of buffer is in use (since all buffers are internally allocated from the same heap).*

---

## Members

`gaBufferSysMem`

Buffer is currently located in system memory

<i>gaBufferCached</i>	Buffer is cached in system memory
<i>gaBufferMoveable</i>	Buffer can be moved around to compact buffer heap
<i>gaBufferPageable</i>	Buffer can be paged to system memory
<i>gaBufferPriority</i>	Buffer is a high priority bitmap
<i>gaBufferNoSysMem</i>	Buffer should never be in system memory
<i>gaBufferFlippable</i>	Buffer is a viewable, flippable surface
<i>gaBuffer3D</i>	Buffer is a hardware 3D capable surface
<i>gaBufferVideo</i>	Buffer is a video overlay window surface
<i>gaBufferDepth</i>	Buffer is a hardware depth buffer
<i>gaBufferTexture</i>	Buffer is a hardware texture map
<i>gaBufferStencil</i>	Buffer is a hardware stencil buffer
<i>gaBufferSpecial</i>	Mask to determine if buffer is special buffer

## GA\_CRTCInfo

### Declaration

```
typedef struct {
    N_uint16    HorizontalTotal;
    N_uint16    HorizontalSyncStart;
    N_uint16    HorizontalSyncEnd;
    N_uint16    VerticalTotal;
    N_uint16    VerticalSyncStart;
    N_uint16    VerticalSyncEnd;
    N_uint32    PixelClock;
    N_uint16    RefreshRate;
    N_uint8     Flags;
} GA_CRTCInfo
```

### Prototype In

snap/graphics.h

### Description

CRTC information block for refresh rate control, passed in to the SetVideoMode function.

The HorizontalTotal, HorizontalSyncStart, HorizontalSyncEnd, VerticalTotal, VerticalSyncStart and VerticalSyncEnd members define the default normalized CRTC values that will be programmed if the gaRefreshCtl flag is passed to SetVideoMode. The CRTC values for a particular resolution will always be the same regardless of color depth. Note also that the CRTC table does not contain any information about the horizontal and vertical blank timing positions. It is up to the driver implementation to determine the correct blank timings to use for the mode when it is initialized depending on the constraints of the underlying hardware (some hardware does not require this information, and most VGA compatible hardware can be very picky about the values programmed for the blank timings).

The Flags member defines the flags that modify the operation of the mode, and the values for this member are defined in the *GA\_CRTCInfoFlagsType* enumeration.

The PixelClock member defines the normalized pixel clock that will be programmed into the hardware. This value is represented in a 32 bit unsigned integer in units of Hz. For example to represent a pixel clock of 25.18Mhz one would code a value of 25,180,000. From the pixel clock and the horizontal and vertical totals, you can calculate the refresh rate for the specific graphics mode using the following formula:

$$\text{refresh rate} = (\text{PixelClock} * 10,000) / (\text{HorizontalTotal} * \text{VerticalTotal})$$

For example a 1024x768 mode with a HTotal of 1360, VTotal of 802, a pixel clock of 130Mhz might be computed as follows:

$$\begin{aligned} \text{refresh rate} &= (130 * 10,000) / (1360 * 802) \\ &= 59.59 \text{ Hz} \end{aligned}$$

The RefreshRate field defines the refresh rate that the CRTC information values define. This value may not actually be used by the driver but must be calculated by the application program using the above formulas before initializing the mode. This entry may be used by the driver to identify any special cases that may need to be handled when setting the mode for specific refresh rates. The value in this field should be represented in units of 0.01 Hz (ie: a value 7200 represents a refresh rate of 72.00Hz).

---

**Note:** *The dwSize member is intended for future compatibility, and should be set to the size of the structure as defined in this header file. Future drivers will be compatible with older software by examining this value.*

---

### Members

<i>HorizontalTotal</i>	Horizontal total (pixels)
<i>HorizontalSyncStart</i>	Horizontal sync start position
<i>HorizontalSyncEnd</i>	Horizontal sync end position
<i>VerticalTotal</i>	Vertical Total (lines)
<i>VerticalSyncStart</i>	Vertical sync start position
<i>VerticalSyncEnd</i>	Vertical sync end position
<i>PixelClock</i>	Pixel clock in units of Hz
<i>RefreshRate</i>	Expected refresh rate in .01Hz
<i>Flags</i>	Initialisation flags for mode

## GA\_CRTCInfoFlagsType

### Declaration

```
typedef enum {
    gaInterlaced      = 0x01,
    gaDoubleScan      = 0x02,
    gaTripleScan      = 0x04,
    gaHSyncNeg        = 0x08,
    gaVSyncNeg        = 0x10
} GA_CRTCInfoFlagsType
```

### Prototype In

snap/graphics.h

### Description

Definitions for flags member of the *GA\_CRTCInfo* structure. These flags define the different flags required to complete a mode set with refresh rate control enabled.

The *gaInterlaced* flag is used to determine whether the mode programmed into the hardware is interlaced or non-interlaced. The CRTC timings passed in will be identical for both interlaced and non-interlaced modes, and it is up to the graphics driver to perform any necessary scaling between the hardware values and the normalized CRTC values in interlaced modes. Note that you must check the *gaHaveInterlaced* bit in the *GA\_modeInfo* structure to determine if interlaced mode is supported before attempting to initialise an interlaced mode.

The *gaDoubleScan* flag is used to determine whether the mode programmed into the hardware is double scanned or not. When double scanning is specified, the vertical CRTC values passed in will be double what the real vertical resolution will be. Double scanning is used to implement the 200, 240 and 300 line graphics modes on modern controllers. Note that you must check the *gaHaveDoubleScan* bit in the *GA\_modeInfo* structure to determine if double scan mode is supported by the hardware in that display mode. Note also that all modes with vertical resolutions below 300 scanline modes require the double scanning to be enabled, and modes between 300 and 400 scanlines can usually look better if it is enabled.

The *gaHSyncNeg* flag is used to determine if the horizontal sync polarity should be set to a negative sync (*gaHSyncNeg* is set) or positive sync (*gaHSyncNeg* is not set).

The *gaVSyncNeg* flag is used to determine if the vertical sync polarity should be set to a negative sync (*gaVSyncNeg* is set) or positive sync (*gaVSyncNeg* is not set).

### Members

<i>gaInterlaced</i>	Enable interlaced mode
<i>gaDoubleScan</i>	Enable double scanned mode
<i>gaTripleScan</i>	Enable triple scanned mode
<i>gaHSyncNeg</i>	Horizontal sync is negative
<i>gaVSyncNeg</i>	Vertical sync is negative



## GA\_CertifyFlagsType

### Declaration

```
typedef enum {
    gaCertified           = 0x01,
    gaCertifiedDDC        = 0x02,
    gaCertifiedStereo     = 0x04,
    gaCertifiedDDC_NT     = 0x08,
    gaCertifiedStereo_NT  = 0x10
} GA_CertifyFlagsType
```

### Prototype In

snap/graphics.h

### Description

Definitions for values stored in the CertifyFlags member of the *GA\_certifyChipInfo* and structure. These flags define what certification tests have been run on the included drivers.

The *gaCertified* value indicates that the driver has been certified to have passed all the base certification tests, and is usually the only important flag for most situations.

The *gaCertifiedDDC* value indicates that the driver has also passed all Windows DDC certification tests, running on the IHV Windows drivers. These tests are independent of the base certification tests as they are dependant on proper testing with the IHV Windows drivers as well as the base certification tests. The *gaCertifiedDDC\_NT* flag is equivalent but indicates that the chipset driver is also certified for DDC on the Windows NT platform.

The *gaCertifiedStereo* value indicates that the driver has also passed all Windows Stereo certification tests, running on the IHV Windows drivers. These tests are independent of the base certification tests as they are dependant on proper testing with the IHV Windows drivers as well as the base certification tests. The *gaCertifiedStereo\_NT* flag is equivalent but indicates that the chipset driver is also certified for stereo on the Windows NT platform.

The *gaCertifiedDFPOutput* value indicates that the driver has also passed all the external digital flat panel tests such that attaching an external digital flat panel to the graphics card will function correctly.

The *gaCertifiedTVOutput* value indicates that the driver has also passed all the external TV output tests such that attaching an external television to either the composite or S-Video connectors of the graphics card will function correctly.

### Members

<i>gaCertified</i>	Driver has passed all base certification tests
<i>gaCertifiedDDC</i>	Driver has passed all Windows DDC tests
<i>gaCertifiedStereo</i>	Driver has passed all Windows stereo tests
<i>gaCertifiedDDC_NT</i>	Driver has passed all Windows NT DDC tests

*gaCertifiedStereo\_NT*

Driver has passed all Windows NT stereo tests

## GA\_DPMSFuncs

**Prototype In**

snap/graphics.h

**Description**

Function group containing all DPMS Display Power Management functions available for the device. These functions are used to power down the external CRT or LCD flat panel monitor via the VESA Display Power Management Specification standard.

---

**Note:** *Be sure to fill in the dwSize member of this structure when you call GA\_queryFunctions to the correct size of the structure at compile time!*

---

## DPMSdetect

Detects the presence of DPMS Power Management features

### Declaration

```
int NAPI_GA_DPMSFuncs::DPMSdetect(  
    N_int16 *supportedStates)
```

### Prototype In

snap/ddc.h

### Parameters

*supportedStates*                      Place to return the supported DPMS states

### Return Value

1 if supported, 0 if not supported.

### Description

Detects what Display Power Management features are provided by the loaded driver. The list of available Power Management states is returned via *supportedStates*, which consist of the flags defined in the `DDC_DPMSFlagsType` enumeration.

### See Also

*DPMSdetect*, *DPMSsetState*

## DPMSsetState

Sets the requested DPMS Power Management state.

### Declaration

```
void NAPI_GA_DPMSFuncs::DPMSsetState(  
    N_int32 state)
```

### Prototype In

snap/ddc.h

### Parameters

*state*                      New DPMS state to set

### Description

This function sets the requested Power Management state. The list of valid states that can be set is defined in the *DDC\_DPMSStructuresType* enumeration.

### See Also

*DPMSdetect*, *DPMSsetState*

## GA\_LCDUseBIOSFlagsType

### Declaration

```
typedef enum {
    gaLCDUseBIOS_Auto    = 0,
    gaLCDUseBIOS_Off     = 1,
    gaLCDUseBIOS_On      = 2
} GA_LCDUseBIOSFlagsType
```

### Prototype In

snap/graphics.h

### Description

Definitions for values stored in the bLCDUseBIOS member of the *GA\_globalOptions* structure. These flags define the different values for this particular option which is multi-state and not just an on/off option.

The gaLCDUseBIOS\_Off value indicates that the the BIOS should not be used for LCD panel modes unless the user has manually switched the driver to run in LCD only or Simultaneous mode. In this mode some laptops will not be able to switch into LCD mode from CRT only mode if the user uses the laptop hot-key switching (the SNAP Graphics API can always be used for correct switching).

The gaLCDUseBIOS\_Auto value will auto select the best option at driver load time. If the system boots up in either LCD only mode or in Simultaneous mode, the driver will always use the BIOS even if the user switches to CRT only mode (ie: equivalent to gaLCDUseBIOS\_On). If the system boot up in CRT only mode, the driver will not use the BIOS unless the user manually switches to LCD panel mode using the SNAP Graphics API (ie: equivalent to gaLCDUseBIOS\_Off).

gaLCDUseBIOS\_On value indicates that the BIOS should always be used for drivers with LCD panel support. This means that even when the user uses the SNAP Graphics API to switch to CRT only mode, the BIOS will be used to set the mode (with restricted modes and features). This mode is useful if the user switches to and from LCD panel and CRT only modes and needs the hot key switching provided in the BIOS to function correctly.

### Members

<i>gaLCDUseBIOS_Auto</i>	Auto select the best option at driver load time
<i>gaLCDUseBIOS_Off</i>	Only use the BIOS modes when running on the LCD
<i>gaLCDUseBIOS_On</i>	Always use the BIOS for drivers with LCD panel support

## GA\_MakeVisibleBufferFlagsType

### Declaration

```
typedef enum {
    gaTripleBuffer          = 0,
    gaWaitVRT               = 1,
    gaDontWait              = 2
} GA_MakeVisibleBufferFlagsType
```

### Prototype In

snap/graphics.h

### Description

Flags passed to the MakeVisibleBuffer function for the waitVRT parameter.

The gaTripleBuffer flag is used to indicate that the visible buffers should be flipped using hardware or software triple buffering where available. This may not be available on all platforms, and if not available gaDontWait is used instead. Hence you may get tearing using this value if the hardware or software triple buffering is not supported and the frame rate of your application is faster than the vertical refresh rate of the display.

The gaWaitVRT flag is used to indicate that the visible buffers should be flipped and that the code should wait for the vertical retrace period before returning. This is necessary to avoid any tearing on the screen if you are doing double buffering, and is the most common value passed to the MakeVisibleBuffer function.

The gaDontWait flag is used to indicate that the visible buffers should be flipped but the code should exit immediately and not wait for the vertical retrace period.

---

**Note:** *If there are only two flippable buffers allocated, the gaTripleBuffer flag will be converted to the gaWaitVRT parameter.*

---

### Members

<i>gaTripleBuffer</i>	Flip the buffers with triple buffering if available
<i>gaWaitVRT</i>	Flip the buffers and wait for vertical retrace
<i>gaDontWait</i>	Flip the buffers and don't wait for retrace

## GA\_OutputFlagsType

### Declaration

```
typedef enum {
    gaOUTPUT_CRT           = 0x0001,
    gaOUTPUT_LCD           = 0x0002,
    gaOUTPUT_DFP           = 0x0400,
    gaOUTPUT_TV            = 0x0004,
    gaOUTPUT_SELECTMASK    = 0x0407,
    gaOUTPUT_TVNTSC        = 0x0008,
    gaOUTPUT_TVNTSC_J      = 0x0010,
    gaOUTPUT_TVPAL         = 0x0020,
    gaOUTPUT_TVPAL_M       = 0x0040,
    gaOUTPUT_TVPAL_60      = 0x0080,
    gaOUTPUT_TVPAL_CN      = 0x0100,
    gaOUTPUT_TVSCART_PAL   = 0x0200,
    gaOUTPUT_TVUNDERSCAN   = 0x0000,
    gaOUTPUT_TVOVERSCAN    = 0x8000,
    gaOUTPUT_TVCOLORMASK   = 0x03F8
} GA_OutputFlagsType
```

### Prototype In

snap/graphics.h

### Description

Flags for the different output displays supported by the driver. These flags are passed to the SetDisplayOutput function to change the currently active display device and the GetDisplayDevice function to determine what devices are currently being used to display output.

---

**Note:** *The color format for TV modes may be specified, or it may not. In the case where the color format is not specified, the currently active color format will be used. In some cases the color format is set in hardware and cannot be changed.*

---

### Members

<i>gaOUTPUT_CRT</i>	Indicates output is sent to CRT display
<i>gaOUTPUT_LCD</i>	Indicates output is sent to LCD panel
<i>gaOUTPUT_DFP</i>	Indicates output is sent to external DFP connector
<i>gaOUTPUT_TV</i>	Indicates output is sent to TV connector
<i>gaOUTPUT_SELECTMASK</i>	Mask to mask out just the output selection
<i>gaOUTPUT_TVNTSC</i>	Set TVOut connector color format to NTSC
<i>gaOUTPUT_TVNTSC_J</i>	Set TVOut connector color format to NTSC-J
<i>gaOUTPUT_TVPAL</i>	Set TVOut connector color format to PAL
<i>gaOUTPUT_TVPAL_M</i>	Set TVOut connector color format to PAL-M
<i>gaOUTPUT_TVPAL_60</i>	Set TVOut connector color format to PAL-60
<i>gaOUTPUT_TVPAL_CN</i>	Set TVOut connector color format to PAL-CN
<i>gaOUTPUT_TVSCART_PAL</i>	Set TVOut connector color format to SCART-PAL
<i>gaOUTPUT_TVUNDERSCAN</i>	Indicates TV output should be underscanned
<i>gaOUTPUT_TVOVERSCAN</i>	Indicates TV output should be overscanned
<i>gaOUTPUT_TVCOLORMASK</i>	Mask to mask out TV color format



## GA\_SCIFuncs

**Prototype In**

snap/graphics.h

**Description**

Function group containing all SCI Serial Control Interface functions available for the device. These functions are used to communicate over the I2C bus with external devices such as DDC (or Plug and Play) monitors, TV encoders and TV tuners.

---

**Note:** *Be sure to fill in the `dwSize` member of this structure when you call `GA_queryFunctions` to the correct size of the structure at compile time!*

---

## SCIbegin

Begin serial communications.

### Declaration

```
void NAPI_GA_SCIFuncs::SCIbegin(  
    N_int32 channel)
```

### Prototype In

snap/ddc.h

### Parameters

*channel*                      Channel to control via I2C (*DDC\_ChannelsType*)

### Description

This function starts a series of SCI communications on the bus and puts the graphics hardware into the necessary state for SCI communications. You *must* call this function before you can call any of the *SCIreadXX* or *SCIwriteXX* functions. You must also call *SCIend* to complete communications when you are done.

### See Also

*SCIdetect*, *SCIbegin*, *SCIwriteSCL*, *SCIwriteSDA*, *SCIreadSCL*, *SCIreadSDA*, *SCIend*

## SCIdetect

Detects the presence of Serial Control Interface functions.

### Declaration

```
int NAPI_GA_SCIFuncs::SCIdetect(
    N_uint8 *capabilities,
    N_int32 *numchannels)
```

### Prototype In

snap/ddc.h

### Parameters

<i>capabilities</i>	Place to store SCI capabilities
<i>numchannels</i>	Place to store number of ports supported

### Return Value

1 if supported, 0 if not supported.

### Description

Detects if the SCI Serial Control Interface extensions are provided by the loaded driver. The capabilities of the SCI interface is returned via the capabilities parameter, which consist of the flags defined in *DDC\_SCIFlagsType*. Generally, application software will not directly control the SCI interface, but will use the higher level DDC and MCS functions, which implement packet based protocols over the SCI interface. The DDC and MCS functions all internally use the SCI interface functions to communicate with the monitor.

### See Also

*SCIdetect*, *SCIbegin*, *SCIwriteSCL*, *SCIwriteSDA*, *SCIreadSCL*, *SCIreadSDA*, *SCIend*

## SClend

End serial communications.

### Declaration

```
void NAPI_GA_SCIFuncs::SClend(  
    N_int32 channel)
```

### Prototype In

snap/ddc.h

### Parameters

*channel*                      Channel to control via I2C (*DDC\_ChannelsType*)

### Description

Complete serial communications over the I2C bus. This function *must* be called after doing communications over the bus.

### See Also

*SCIdetect, SClbegin, SClwriteSCL, SClwriteSDA, SClreadSCL, SClreadSDA, SClend*

## SCIreadSCL

Reads the SCL clock line

### Declaration

```
int NAPI_GA_SCIFuncs::SCIreadSCL(  
    N_int32 channel)
```

### Prototype In

snap/ddc.h

### Parameters

*channel*                      Channel to control via I2C (*DDC\_ChannelsType*)

### Return Value

Bit value read from port

### Description

This function reads the value from the SCL clock line and returns it. Note that not all controllers support this function so you must determine if this function is supported by looking at the returned SCI capabilities flags. For cards that do not allow reading of the clock line, a delay based system must be used to slow down the host during transfer to the display. Normally the SCL line is used by the display device to speed throttle the host transfers by forcing the SCL line low while it is busy and unable to respond.

### See Also

*SCIdetect, SCIbegin, SCIwriteSCL, SCIwriteSDA, SCIreadSCL, SCIreadSDA, SCIend*

## SCIreadSDA

Reads the SDA data line

### Declaration

```
int NAPI_GA_SCIFuncs::SCIreadSDA(  
    N_int32 channel)
```

### Prototype In

snap/ddc.h

### Parameters

*channel*                      Channel to control via I2C (*DDC\_ChannelsType*)

### Return Value

Bit value read from port

### Description

This function read the value from the SDA data line and returns it.

### See Also

*SCIdetect, SCIbegin, SCIwriteSCL, SCIwriteSDA, SCIreadSCL, SCIreadSDA, SCIend*

## SCIwriteSCL

Sets the SCL clock line to the specified value

### Declaration

```
void NAPI_GA_SCIFuncs::SCIwriteSCL(  
    N_int32 channel,  
    N_int32 bit)
```

### Prototype In

snap/ddc.h

### Parameters

<i>channel</i>	Channel to control via I2C ( <i>DDC_ChannelsType</i> )
<i>bit</i>	Bit value to write to port (0 or 1)

### Description

This function sets the SCL clock line to the specified value

### See Also

*SCIdetect, SCIdbegin, SCIwriteSCL, SCIwriteSDA, SCReadSCL, SCReadSDA, SCIdend*

## SCIwriteSDA

Sets the SDA data line to the specified value

### Declaration

```
void NAPI_GA_SCIFuncs::SCIwriteSDA(  
    N_int32 channel,  
    N_int32 bit)
```

### Prototype In

snap/ddc.h

### Parameters

<i>channel</i>	Channel to control via I2C ( <i>DDC_ChannelsType</i> )
<i>bit</i>	Bit value to write to port (0 or 1)

### Description

This function sets the SDA data line to the specified value

### See Also

*SCIdetect, SCIbegin, SCIwriteSCL, SCIwriteSDA, SCIreadSCL, SCIreadSDA, SCIend*



## GA\_TVParams

### Declaration

```
typedef struct {  
    N_int16      hPos;  
    N_int16      vPos;  
    N_uint16     brightness;  
    N_uint16     contrast;  
} GA_TVParams
```

### Prototype In

snap/graphics.h

### Description

Structure used to describe the TV parameters specific to a particular TV output mode. We store these values independently in the options structure for different TV modes (ie: 640x480, 800x600, PAL, NTSC etc).

### Members

<i>hPos</i>	Horizontal position value (+-)
<i>vPos</i>	Vertical position value (+-)
<i>brightness</i>	Brightness control value
<i>contrast</i>	Contrast control value

## GA\_VBEFuncs

**Prototype In**

snap/graphics.h

**Description**

Function group containing all VBE/Core emulation functions. These functions should generally not be used by application programs directly, as they are only intended to be used by the VBE/Core emulation driver.

---

**Note:** *Be sure to fill in the `dwSize` member of this structure when you call `GA_queryFunctions` to the correct size of the structure at compile time!*

---

## GetPaletteData

Returns the current hardware color palette in VBE/Core compatible mode

### Declaration

```
void GA_VBEFuncs::GetPaletteData(
    GA_palette *pal,
    N_int32 num,
    N_int32 index)
```

### Prototype In

snap/graphics.h

### Parameters

<i>pal</i>	Place to return the palette data
<i>num</i>	Number of palette entries to read
<i>index</i>	Index of first entry to read

### Description

This function is similar to *GetPaletteData*, however the palette data is always read verbatim, and no palette translation occurs. Hence these functions are compatible with the way the VESA VBE/Core palette functions work, which is different to the native SNAP way of handling palette programming.

### See Also

*SetBytesPerLine*, *Set8BitDAC*, *SetPaletteData*

## Set8BitDAC

Allow the RAMDAC width to be changed at runtime

### Declaration

```
ibool GA_VBEFuncs::Set8BitDAC(  
    ibool enable)
```

### Prototype In

snap/graphics.h

### Return Value

True if request fulfilled, false if not

### Description

This function allows the RAMDAC pixel width to be changed between 8-bit per primary mode or the VGA compatible 6-bit per primary mode. This primary use of this function is to allow emulation of VESA VBE/Core functions on top of native SNAP drivers (ie: DOS box support in Windows etc).

### See Also

*SetBytesPerLine, SetPaletteData, GetPaletteData*

## SetBytesPerLine

Allows the pixel stride of the display mode to be changed

### Declaration

```
ibool GA_VBEFuncs::SetBytesPerLine(  
    int bytesPerLine,  
    int *newBytes)
```

### Prototype In

snap/graphics.h

### Parameters

<i>bytesPerLine</i>	Requested scanline pitch to set
<i>newBytes</i>	Place to store actual scanline pitch set

### Description

This function allows the display stride to be changed after the mode has been initialised. Note that this will **only** work reliably if the mode was initialised with the gaNoAccel flag, to ensure that the hardware accelerator is not active. This primary use of this function is to allow emulation of VESA VBE/Core functions on top of native SNAP drivers (ie: DOS box support in Windows etc).

### See Also

*Set8BitDAC, SetPaletteData, GetPaletteData*

## SetPaletteData

Programs the hardware color palette in VBE/Core compatible mode

### Declaration

```
void GA_VBEFuncs::SetPaletteData(
    GA_palette *pal,
    N_int32 num,
    N_int32 index,
    N_int32 waitVRT)
```

### Prototype In

snap/graphics.h

### Parameters

<i>pal</i>	Pointer to the palette data to program
<i>num</i>	Number of palette entries to program
<i>index</i>	Index of first entry to program
<i>waitVRT</i>	Wait for vertical retrace flag

### Description

This function is similar to *SetPaletteData*, however the palette data is always programmed into the hardware palette verbatim, and no palette translation occurs. Hence these functions are compatible with the way the VESA VBE/Core palette functions work, which is different to the native SNAP way of handling palette programming.

### See Also

*SetBytesPerLine*, *Set8BitDAC*, *GetPaletteData*

## GA\_VideoBufferFormatsType

### Declaration

```
typedef enum {
    gaVideoRGB332          = 0x00000001,
    gaVideoRGB555          = 0x00000002,
    gaVideoRGB565          = 0x00000004,
    gaVideoRGB888          = 0x00000008,
    gaVideoRGB8888         = 0x00000010,
    gaVideoYUV9            = 0x00000020,
    gaVideoYUV12           = 0x00000040,
    gaVideoYUV411          = 0x00000080,
    gaVideoYUV422          = 0x00000100,
    gaVideoYUV444          = 0x00000200,
    gaVideoYCrCb422        = 0x00000400,
    gaVideoYUYV            = 0x08000000,
    gaVideoYVYU            = 0x10000000,
    gaVideoUYVY            = 0x20000000,
    gaVideoVYUY            = 0x40000000
} GA_VideoBufferFormatsType
```

### Prototype In

snap/graphics.h

### Description

Flags for hardware video input formats defined in the VideoInputFormats member of the *GA\_videoInf* structure. These flags define the hardware video capabilities of the particular video overlay window, and are only valid if the gaHaveAccelVideo flag is defined in the Attributes member of the *GA\_modelInfo* structure.

The gaVideoRGB332 flag is used to determine whether the video overlay window can support hardware video playback of frames stored in the RGB 3:3:2 format (8 bits per pixel).

The gaVideoRGB555 flag is used to determine whether the video overlay window can support hardware video playback of frames stored in the RGB 5:5:5 format (16 bits per pixel, 1 ignored).

The gaVideoRGB565 flag is used to determine whether the video overlay window can support hardware video playback of frames stored in the RGB 5:6:5 format (16 bits per pixel).

The gaVideoRGB888 flag is used to determine whether the video overlay window can support hardware video playback of frames stored in the RGB 8:8:8 format (24 bits per pixel). Only one RGB format is supported, and the Blue byte is always stored first in memory (ie: B:G:R).

The gaVideoRGB8888 flag is used to determine whether the video overlay window can support hardware video playback of frames stored in the RGB 8:8:8:8 format (32 bits per pixel). Only one RGB format is supported, and the Blue byte is always stored first in memory (ie: B:G:R:A).

The `gaVideoYUV9` flag is used to determine whether the video overlay window can support hardware video playback of frames stored in the YUV9 format. For more information on different YUV formats and how they are actually stored in the framebuffer, see the section titled 'Overview of YUV pixels'.

The `gaVideoYUV12` flag is used to determine whether the video overlay window can support hardware video playback of frames stored in the YUV12 format. For more information on different YUV formats and how they are actually stored in the framebuffer, see the section titled 'Overview of YUV pixels'.

The `gaVideoYUV411` flag is used to determine whether the video overlay window can support hardware video playback of frames stored in the YUV 4:1:1 format. The YUV 4:1:1 data can be stored with the YUV values in varying formats, and you should check the `gaVideoYUYV` etc flags to determine which formats are supported by this controller. For more information on different YUV formats and how they are actually stored in the framebuffer, see the section titled 'Overview of YUV pixels'.

The `gaVideoYUV422` flag is used to determine whether the video overlay window can support hardware video playback of frames stored in the YUV 4:2:2 format. The YUV 4:2:2 data can be stored with the YUV values in varying formats, and you should check the `gaVideoYUYV` etc flags to determine which formats are supported by this controller. For more information on different YUV formats and how they are actually stored in the framebuffer, see the section titled 'Overview of YUV pixels'.

The `gaVideoYUV444` flag is used to determine whether the video overlay window can support hardware video playback of frames stored in the YUV 4:4:4 format. The YUV 4:4:4 data can be stored with the YUV values in varying formats, and you should check the `gaVideoYUYV` etc flags to determine which formats are supported by this controller. For more information on different YUV formats and how they are actually stored in the framebuffer, see the section titled 'Overview of YUV pixels'.

The `gaVideoYCrCb422` flag is used to determine whether the video overlay window can support hardware video playback of frames stored in the YCrCb 4:2:2 format. The YCrCb 4:2:2 data can be stored with the YUV values in varying formats, and you should check the `gaVideoYUYV` etc flags to determine which formats are supported by this controller. For more information on different YUV formats and how they are actually stored in the framebuffer, see the section titled 'Overview of YUV pixels'.

The `gaVideoYUYV`, `gaVideoYVYU`, `gaVideoUYVY` and `gaVideoVYUY` flags are used to determine what YUV pixel layouts is supported for the above supported YUV pixel formats.

---

**Note:** *These flags are also passed to the `AllocVideoBuffer` function to determine the video input data type being displayed for the video window.*

---



---

**Note:** The `gaVideoYUYV` and related flags define the YUV pixel layouts that are supported by the hardware for the YUV input format it supports. For instance the hardware may report `gaVideoYUV422` and the `gaVideoYUYV` flags, which means it supports the YUV422 format with the format 4:2:4:2 (Y:U:Y:V) in video memory. See the section titled 'Overview of YUV Pixels' for more information.

---

### Members

<code>gaVideoRGB332</code>	Supports RGB 3:3:2 input format
<code>gaVideoRGB555</code>	Supports RGB 5:5:5 input format
<code>gaVideoRGB565</code>	Supports RGB 5:6:5 input format
<code>gaVideoRGB888</code>	Supports RGB 8:8:8 input format
<code>gaVideoRGB8888</code>	Supports RGB 8:8:8:8 input format
<code>gaVideoYUV9</code>	Supports YUV9 input format
<code>gaVideoYUV12</code>	Supports YUV12 input format
<code>gaVideoYUV411</code>	Supports YUV411 input format
<code>gaVideoYUV422</code>	Supports YUV422 input format
<code>gaVideoYUV444</code>	Supports YUV444 input format
<code>gaVideoYCrCb422</code>	Supports YCrCb422 input format
<code>gaVideoYUYV</code>	Supports the YUYV pixel layout (for the above YUV formats)
<code>gaVideoYVYU</code>	Supports the YVYU pixel layout (for the above YUV formats)
<code>gaVideoUYVY</code>	Supports the UYVY pixel layout (for the above YUV formats)
<code>gaVideoVYUY</code>	Supports the VYUY pixel layout (for the above YUV formats)

## GA\_VideoOutputFlagsType

### Declaration

```
typedef enum {
    gaVideoXInterp          = 0x00000001,
    gaVideoYInterp          = 0x00000002,
    gaVideoColorKeySrcSingle = 0x00000004,
    gaVideoColorKeySrcRange  = 0x00000008,
    gaVideoColorKeyDstSingle = 0x00000010,
    gaVideoColorKeyDstRange  = 0x00000020
} GA_VideoOutputFlagsType
```

### Prototype In

snap/graphics.h

### Description

Flags for hardware video output capabilities defined in the VideoOutputFlags member of the *GA\_videoInf* structure. These flags define the hardware video capabilities of the particular graphics mode, and are only valid if the gaHaveAccelVideo flag is defined in the Attributes member of the *GA\_modelInfo* structure.

The gaVideoXInterp flag is used to determine whether the video overlay window can support hardware interpolation or filtering in the X axis when scaling the input image to the display. If this bit is 1, then the hardware can support filtering of values in the X direction resulting in better looking images when scaled from a smaller input frame.

The gaVideoYInterp flag is used to determine whether the video overlay window can support hardware interpolation or filtering in the Y axis when scaling the input image to the display. If this bit is 1, then the hardware can support filtering of values in the Y direction resulting in better looking images when scaled from a smaller input frame.

The gaVideoColorKeySrcSingle flag is used to determine whether the video overlay window can support hardware source color keying with a single source color key value. When hardware source color keying is enabled, any pixel data in the incoming source video that matches the currently set video color key will be ignored and not displayed on the screen, essentially allowing the display data under the video overlay window to show through.

The gaVideoColorKeySrcRange flag is used to determine whether the video overlay window can support hardware source color keying with a range of color key values. This is the same as single source color keying, but the color key values may be allowed to fall within a range of available colors (useful if data has been filtered causing the colors to change).

The gaVideoColorKeyDstSingle flag is used to determine whether the video overlay window can support hardware destination color keying with a single destination color key value. When hardware destination color keying is enabled (sometimes called blue-screening), any destination pixels that the overlay window overlaps that match the

currently set video color key, will cause the source input pixels to be ignored, essentially allowing the display data under the video overlay window to show through.

The `gaVideoColorKeyDstRange` flag is used to determine whether the video overlay window can support hardware destination color keying with a range of color key values. This is the same as single destination color keying, but the color key values may be allowed to fall within a range of available colors.

---

**Note:** *These flags are also passed to the `SetVideoOutput` function to determine what features are enabled for the output window.*

---

### Members

<code>gaVideoXInterp</code>	Supports X interpolation
<code>gaVideoYInterp</code>	Supports Y interpolation
<code>gaVideoColorKeySrcSingle</code>	Supports source color keying, single color
<code>gaVideoColorKeySrcRange</code>	Supports source color keying, range of colors
<code>gaVideoColorKeyDstSingle</code>	Support destination color keying, single color
<code>gaVideoColorKeyDstRange</code>	Support destination color keying, range of colors

## GA\_WorkAroundsFlagsType

### Declaration

```
typedef enum {
    gaSlowBltSys                = 0x00000001,
    gaHWCursor32x32            = 0x00000002,
    gaHWCursorBlackBackground  = 0x00000004,
    gaSlow24bpp                = 0x00000008,
    gaSlow32bpp                = 0x00000010,
    gaBrokenLines              = 0x00000020,
    gaNoDDCBIOS                = 0x00000040,
    gaNoWriteCombine           = 0x00000080,
    gaNoInterlacedCursor       = 0x00000100,
    gaHWCursorBlackAndWhite8bpp = 0x00000200,
    gaNoLCDSwitching           = 0x00000400,
    gaNoLCDExpandCursor        = 0x00000800,
    gaUsesBIOS                 = 0x00001000,
    gaNeedFullBIOS             = 0x00002000,
    gaNeedContiguousFlipBuffers = 0x00004000,
    gaNeed3DTiledAddressing     = 0x00008000,
    gaNoWHQLTransparentBlit     = 0x00010000
} GA_WorkAroundsFlagsType
```

### Prototype In

snap/graphics.h

### Description

Flags for the WorkArounds member of the *GA\_devCtx* structure. These flags define conditions for uncommon hardware bugs that can't easily be handled via the generic SNAP Graphics information reporting mechanism. Any code that calls the SNAP Graphics hardware drivers directly must be aware of these workarounds and how to deal with them. However the SNAP Graphics Software Reference Rasteriser knows how to deal with all currently known bugs, so application developers should use the reference rasteriser at all times for maximum compatibility with new hardware drivers.

The *gaSlowBltSys* flag indicates that the hardware *BitBltSys* function is faster than a direct linear framebuffer software blit. Most modern hardware can do a software blit as fast or faster than using the hardware, but some hardware can be faster than a pure software blit. This is only true when the mix mode is *GA\_REPLACE\_MIX*, since software reads from the framebuffer over the PCI bus are terribly slow.

The *gaHWCursor32x32* flag indicates that the hardware only supports a 32x32 hardware cursor, while the SNAP Graphics specification implements an interface for 64x64 hardware cursors. SNAP Graphics drivers will still implement hardware cursor support for hardware that only supports a 32x32 cursor, however this flag will be set. If the high level operating system drivers require a cursor larger than 32x32, then this flag should be checked and a software cursor used in it's place when this is the case.

The *gaSlow24bpp* flag indicates that although the 24bpp display modes are accelerated, they are only partially accelerated. Hence if there is an equivalent 32bpp display mode, that most should be used in preference to the 24bpp display mode if possible.

The `gaSlow32bpp` flag indicates that although the 32bpp display modes are accelerated, they are only partially accelerated. Hence if there is an equivalent 24bpp display mode, that most should be used in preference to the 32bpp display mode if possible.

The `gaBrokenLines` flag indicates that the hardware line drawing produces slightly different pixels than the software reference rasterizer and cannot be made to produce correct pixels. For this reason, conformance testing for line drawing will be skipped on this hardware.

The `gaNoDDCBIOS` flag is an internal flag to indicate that the card does not have DDC BIOS support, and hence we should not attempt to use the DDC BIOS functions to read the EDID for legacy devices.

The `gaNoWriteCombine` flag is an internal flag to indicate that the graphics chipset does not work properly when write combining is enabled for later processors. If write combining is used, it may cause the system to lock or hang.

The `gaNoInterlacedCursor` flag is an internal flag to indicate that the graphics chipset does not properly handle hardware cursors in interlaced display modes. Hence a software cursor should be used instead for these modes.

The `gaNoLCDSwitching` flag is set if the `SetOptions` function does not properly implement LCD/CRT switching. This is usually set for situations where the BIOS is not working properly, and it will be up to the user to use the function keys on the laptop to do the switching.

The `gaNoLCDExpandCursor` flag is set if the hardware does not support the hardware cursor correctly in LCD panel expansion modes. Hence a software mouse cursor should be used instead.

The `gaUsesBIOS` flag is set if the driver internally uses the BIOS for mode sets. This is mostly a flag to let the OS/2 driver know that it does not need to implement BIOS specific hacks that can slow down mode switching as the BIOS is already being used internally in the drivers. The BIOS is generally only used for laptop support and for legacy drivers where there is not enough information available to work without the BIOS.

## Members

<i><code>gaSlowBltSys</code></i>	Software is slower than hardware for <code>GA_REPLACE_MODE</code>
<i><code>gaHWCursor32x32</code></i>	The hardware cursor is only 32x32 in size
<i><code>gaHWCursorBlackBackground</code></i>	The hardware cursor requires that the background color always be black (0's)
<i><code>gaSlow24bpp</code></i>	The 24bpp modes are only partially accelerated
<i><code>gaSlow32bpp</code></i>	The 32bpp modes are only partially accelerated
<i><code>gaBrokenLines</code></i>	The hardware line drawing is not

<i>gaNoDDCBIOS</i>	conformant Card does not have DDC BIOS support
<i>gaNoWriteCombine</i>	Card does not support write combining
<i>gaNoInterlacedCursor</i>	HW cursor in interlaced modes is broken
<i>gaHWCursorBlackAndWhite8bpp</i>	The hardware cursor in 8bpp is always black and white and cannot be changed.
<i>gaNoLCDSwitching</i>	This flag is set if LCD switching does not work
<i>gaNoLCDEExpandCursor</i>	This flag is set to disable cursor in LCD expand modes
<i>gaUsesBIOS</i>	Internally this driver uses the BIOS
<i>gaNeedFullBIOS</i>	This driver needs a full BIOS implementation on OS/2 in order to function
<i>gaNeedContiguousFlipBuffers</i>	Contiguous flip buffers needed for Win32 DirectDraw
<i>gaNeed3DTiledAddressing</i>	3D tiled addressing needed for Win32 DirectDraw
<i>gaNoWHQLTransparentBlit</i>	Chipset does not support WHQL style transparent blits.

## GA\_blendFuncType

### Declaration

```
typedef enum {
    gaBlendNone,
    gaBlendZero,
    gaBlendOne,
    gaBlendSrcColor,
    gaBlendOneMinusSrcColor,
    gaBlendSrcAlpha,
    gaBlendOneMinusSrcAlpha,
    gaBlendDstAlpha,
    gaBlendOneMinusDstAlpha,
    gaBlendDstColor,
    gaBlendOneMinusDstColor,
    gaBlendSrcAlphaSaturate,
    gaBlendConstantColor,
    gaBlendOneMinusConstantColor,
    gaBlendConstantAlpha,
    gaBlendOneMinusConstantAlpha,
    gaBlendSrcAlphaFast,
    gaBlendConstantAlphaFast
} GA_blendFuncType
```

### Prototype In

snap/graphics.h

### Description

Flags for 2D alpha blending functions supported by the SNAP Graphics drivers. The values in here define the the alpha blending functions passed to the *srcBlendFunc* and *dstBlendFunc* parameters of the *SetBlendFunc* function. Essentially the blend function defines how to combine the source and destination pixel color together to get the resulting destination color during rendering. The formula used for this is defined as:

$$\text{DstColor} = \text{SrcColor} * \text{SrcFunc} + \text{DstColor} * \text{DstFunc};$$

If the source alpha blending function is set to *gaBlendConstantAlpha*, the *SrcFunc* above becomes:

$$\text{SrcFunc} = \text{ConstAlpha}$$

If the destination alpha blending function is set to *gaBlendOneMinusDstAlpha* then *DstFunc* above becomes:

$$\text{DstFunc} = (1 - \text{DstAlpha})$$

and the final equation becomes (note that each color channel is multiplied individually):

$$\text{DstColor} = \text{SrcColor} * \text{ConstAlpha} + \text{DstColor} * (1 - \text{DstAlpha})$$

Although the above is a completely contrived example, it does illustrate how the functions defined below combine to allow you to build complex and interesting blending functions. For simple source alpha transparency, the following formula would usually be used:

```
DstColor = SrcColor * SrcAlpha + DstColor * (1-SrcAlpha)
```

If you wish to use this type of blending and you do not care about the resulting alpha channel information, you can set the optimised `gaBlendSrcAlphaFast` blending mode. If you set both the source and destination blending modes to this value, the above formula will be used but an optimised fast path will be taken internally to make this run as fast as possible. For normal blending operations this will be much faster than setting the above formula manually. If however you need the destination alpha to be preserved, you will need to use the slower method instead.

For simple constant alpha transparency, the following formula would usually be used:

```
DstColor = SrcColor * ConstantAlpha + DstColor * (1-ConstantAlpha)
```

If you wish to use this type of blending and you do not care about the resulting alpha channel information, you can set the optimised `gaBlendConstantAlphaFast` blending mode. If you set both the source and destination blending modes to this value, the above formula will be used but an optimised fast path will be taken internally to make this run as fast as possible. For normal blending operations this will be much faster than setting the above formula manually. If however you need the destination alpha to be preserved, you will need to use the slower method instead.

---

**Note:** *All the above equations assume the color values and alpha values are in the range of 0 through 1 in floating point. In reality all blending is done with integer color and alpha components in the range of 0 to 255, when a value of 255 corresponds to a value of 1.0 in the above equations.*

**Note:** *The constant color value set by a call to `SetForeColor`, and the constant alpha value set by a call to `SetAlphaValue`.*

**Note:** *Setting a blending function that uses the destination alpha components is only supported if the framebuffer currently supports destination alpha. Likewise setting a blending function that uses source alpha components is only supported if the framebuffer or incoming bitmap data contains an alpha channel. The results are undefined if these conditions are not met.*

**Note:** *Enabling source or destination alpha blending overrides the setting of the current mix mode. Logical mix modes and blending cannot be used at the same time.*

---

## Members

<code>gaBlendNone</code>	No alpha blending
<code>gaBlendZero</code>	Blend factor is always zero
<code>gaBlendOne</code>	Blend factor is always one
<code>gaBlendSrcColor</code>	Blend factor is source color
<code>gaBlendOneMinusSrcColor</code>	Blend factor is 1-source color
<code>gaBlendSrcAlpha</code>	Blend factor is source alpha
<code>gaBlendOneMinusSrcAlpha</code>	Blend factor is 1-source alpha
<code>gaBlendDstAlpha</code>	Blend factor is destination alpha
<code>gaBlendOneMinusDstAlpha</code>	Blend factor is 1-destination alpha



<i>gaBlendDstColor</i>	Blend factor is destination color
<i>gaBlendOneMinusDstColor</i>	Blend factor is 1-destination color
<i>gaBlendSrcAlphaSaturate</i>	Blend factor is src alpha saturation
<i>gaBlendConstantColor</i>	Blend factor is a constant color
<i>gaBlendOneMinusConstantColor</i>	Blend factor is 1-constant color
<i>gaBlendConstantAlpha</i>	Blend factor is constant alpha
<i>gaBlendOneMinusConstantAlpha</i>	Blend factor is 1-constant alpha
<i>gaBlendSrcAlphaFast</i>	Common case of optimised src alpha
<i>gaBlendConstantAlphaFast</i>	Common case of optimised constant alpha

## GA\_bltFx

### Declaration

```
typedef struct {
    N_uint32      dwSize;
    N_uint32      Flags;
    N_int32       Mix;
    GA_color      ColorKeyLo;
    GA_color      ColorKeyHi;
    N_int32       SrcBlendFunc;
    N_int32       DstBlendFunc;
    GA_color      ConstColor;
    N_uint32      ConstAlpha;
    N_int32       BitsPerPixel;
    GA_pixelFormat *PixelFormat;
    GA_palette     *DstPalette;
    GA_palette     *SrcPalette;
    GA_color       *TranslateVec;
    N_int32       ClipLeft;
    N_int32       ClipTop;
    N_int32       ClipRight;
    N_int32       ClipBottom;
} GA_bltFx
```

### Prototype In

snap/graphics.h

### Description

Hardware 2D BitBltFx information structure. This structure defines the type of BitBlt operation that is performed by the BitBltFx family of functions. The Flags member defines the type of BitBlt operation to be performed, and can be any combination of the supported flags (be sure to call BitBltFxTest first to determine if that combination of effects is supported).

If mixes are enabled, the Mix member is used to determine the mix operation to apply. If mixes are not enabled, GA\_REPLACE\_MIX is assumed (some hardware may not support mix operations for effects blits).

The ColorKeyLo and ColorKeyHi members define the color key ranges if range based color keying is selected. If only a single color key is enabled, the ColorKeyLo value is the value used as the color key. The ColorKeyHi value is inclusive in that it is included in the color range.

If blending is enabled, the SrcBlendFunc, DstBlendFunc and Alpha values are used to implement the blending operation.

If clipping is enabled, the destination clip rectangle is passed in the ClipLeft, ClipTop, ClipRight and ClipBottom members. Clipping is most useful for stretching operations, where clipping in software is problematic.

If color conversion is enabled and you are color converting between color index pixel formats and other pixel formats (including palette remapping), you can optionally pass

in a pre-computed translation vector in the TranslateVec member. This will be used in place of dynamically computing the color translation vector on the fly during the blit operation, so it is faster in cases where the translation vector is known in advance for a number of blit operations.

---

**Note:** *The dwSize member is intended for future compatibility, and should be set to the size of the structure as defined in this header file. Future drivers will be compatible with older software by examining this value.*

**Note:** *The ColorKeyLo and ColorKeyHi values are always color values in the format of the destination surface color depth and pixel format. I.e: if you are color converting an 8bpp bitmap to a 32bpp destination surface, the color key values will be 32bpp color key values, not 8bpp color key values.*

---

## Members

<i>dwSize</i>	Set to size of structure in bytes
<i>Flags</i>	BitBltFx flags to define the type of BitBlt operation ( <i>GA_BitBltFxFlagsType</i> )
<i>Mix</i>	Logical mix operation (if mixes enabled)
<i>ColorKeyLo</i>	Color key low value of range (if color keying enabled)
<i>ColorKeyHi</i>	Color key high value of range (if color keying enabled)
<i>SrcBlendFunc</i>	Src blend function ( <i>GA_blendFuncType</i> )
<i>DstBlendFunc</i>	Dst blend function ( <i>GA_blendFuncType</i> )
<i>ConstColor</i>	Constant color value for blending if blending enabled
<i>ConstAlpha</i>	Constant alpha blend factor (0-255 if blending enabled)
<i>BitsPerPixel</i>	Color depth for the source bitmap
<i>PixelFormat</i>	Pixel format for the source bitmap
<i>DstPalette</i>	Color index palette for destination (if destination color index)
<i>SrcPalette</i>	Color index palette for source bitmap (if source color index)
<i>TranslateVec</i>	Pre-computed color translation vector for color conversion
<i>ClipLeft</i>	Left coordinate for destination clip rectangle
<i>ClipTop</i>	Top coordinate for destination clip rectangle
<i>ClipRight</i>	Right coordinate for destination clip rectangle
<i>ClipBottom</i>	Bottom coordinate for destination clip rectangle

## GA\_buf

### Declaration

```
typedef struct {
    N_uint32    dwSize;
    N_int32     Width;
    N_int32     Height;
    N_int32     Stride;
    N_int32     CacheStride;
    N_int32     StartX;
    N_int32     StartY;
    N_int32     Offset;
    N_int32     Flags;
    N_int32     Format;
    N_int32     UsageCount;
    void        *Surface;
    void        *SurfaceCache;
    void        *AppInfo;
} GA_buf
```

### Prototype In

snap/graphics.h

### Description

Generic offscreen managed buffer structure, which is used to describe offscreen managed buffers, and is allocated using the AllocBuffer function. Offscreen managed buffers are used to allocate and manage offscreen video memory and system memory buffers.

---

**Note:** If the buffer is linear based, the StartX and StartY members will contain a value of -1.

**Note:** The dwSize member is intended for future compatibility, and should be set to the size of the structure as defined in this header file. Future drivers will be compatible with older software by examining this value.

---

### Members

<i>dwSize</i>	Set to size of structure in bytes
<i>Width</i>	Width of buffer in pixels
<i>Height</i>	Height of the buffer in pixels
<i>Stride</i>	Stride of the buffer in bytes (bytes for a line of data)
<i>CacheStride</i>	Stride of the buffer in system memory buffer cache
<i>StartX</i>	Starting X coordinate in framebuffer for buffer (if x,y based)
<i>StartY</i>	Starting Y coordinate in framebuffer for buffer (if x,y based)
<i>Offset</i>	Linear buffer starting address in bytes
<i>Flags</i>	Flags for the buffer ( <i>GA_BufferFlagsType</i> )
<i>Format</i>	Internal format indicator for the buffer
<i>UsageCount</i>	Usage count for tracking pageable buffers
<i>Surface</i>	Pointer to start of the buffer surface
<i>SurfaceCache</i>	Pointer to surface cache in system memory (NULL if

*AppInfo* uncached)  
Pointer to application data if necessary

## GA\_buffer

### Declaration

```
typedef struct {
    N_uint32    dwSize;
    N_int32     Offset;
    N_int32     Stride;
    N_int32     Width;
    N_int32     Height;
} GA_buffer
```

### Prototype In

snap/graphics.h

### Description

Generic graphics buffer parameter block. This structure defines a generic buffer in offscreen video memory, and is passed to the driver to make such buffers active rendering operations. The Offset member is the offset of the start of the buffer in video memory. The Stride member defines the stride of the buffer in bytes, while the Width and Height members define the dimensions of the buffer in logical pixel units.

---

**Note:** *All buffers are in packed pixel format, and the values of the Offset and Stride members must adhere to the format restrictions defined in the GA\_modelInfo structure for the buffer type being enabled.*

**Note:** *The dwSize member is intended for future compatibility, and should be set to the size of the structure as defined in this header file. Future drivers will be compatible with older software by examining this value.*

---

### Members

<i>dwSize</i>	Set to size of structure in bytes
<i>Offset</i>	Buffer starting address in bytes
<i>Stride</i>	Stride of the buffer in bytes (bytes for a line of data)
<i>Width</i>	Width of buffer in pixels
<i>Height</i>	Height of the buffer in pixels

## GA\_bufferFuncs

**Prototype In**

snap/graphics.h

**Description**

Function group containing all offscreen buffer management function available via the SNAP API's. This function group is *only* returned by the 2D reference rasteriser library, and not by hardware drivers.

---

**Note:** *Be sure to fill in the dwSize member of this structure when you call GA\_queryFunctions to the correct size of the structure at compile time!*

---

## AllocBuffer

Allocate a buffer with the requested attributes

### Declaration

```
GA_buf * GA_bufferFuncs::AllocBuffer(
    N_int32 width,
    N_int32 height,
    N_int32 flags)
```

### Prototype In

snap/graphics.h

### Parameters

<i>width</i>	Width of the buffer in pixels
<i>height</i>	Height of the buffer in scanlines
<i>flags</i>	Flags used to create the buffer ( <i>GA_BufferFlagsType</i> )

### Return Value

Pointer to the allocated buffer, NULL on failure.

### Description

This function allocates a new buffer and returns it. This only allocates video memory buffers for drawing and bitmap storage and caching. It is not used to allocate depth buffers or texture buffers.

SEE ALSO *FreeBuffer*, *GetPrimaryBuffer*, *FlipToBuffer*, *GetFlippableBuffer*, *LockBuffer*, *BitBlTBuf*, *SetActiveBuffer*



## BitBltBuf

Copies pixels from one buffer into the currently active buffer

### Declaration

```
void GA_bufferFuncs::BitBltBuf(
    GA_buf *buf,
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 width,
    N_int32 height,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 mix)
```

### Prototype In

snap/graphics.h

### Parameters

<i>buf</i>	SNAP buffer to blit to the active surface
<i>srcLeft</i>	Left coordinate of the source rectangle to copy
<i>srcTop</i>	Top coordinate of the source rectangle to copy
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>mix</i>	Mix code for the copy ( <i>GA_mixCodesType</i> )

### Description

This function copies a rectangular region from the source buffer to the active drawing surface, copying from (srcLeft, srcTop, srcLeft+width-1, srcTop+height-1) in the source buffer to (dstLeft, dstTop) in the active buffer. The specified mix is used to combine the pixels in the active buffer. This function will also correctly handle cases of overlapping regions if the source buffer is the same as the active buffer.

### See Also

*BitBltPattBuf*, *BitBltColorPattBuf*, *SrcTransBltBuf*, *DstTransBltBuf*, *BitBltPlaneMaskedBuf*, *BitBltFxBuf*, *DrawRectBuf*

## BitBltColorPattBuf

Copies pixels from one buffer into the currently active buffer, applying a color 8x8 pattern

### Declaration

```
void GA_bufferFuncs::BitBltColorPattBuf(
    GA_buf *buf,
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 width,
    N_int32 height,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 rop3)
```

### Prototype In

snap/graphics.h

### Parameters

<i>buf</i>	SNAP buffer to blit to the active surface
<i>srcLeft</i>	Left coordinate of the source rectangle to copy
<i>srcTop</i>	Top coordinate of the source rectangle to copy
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>rop3</i>	ROP3 code for the copy ( <i>GA_rop3CodesType</i> )

### Description

This function copies a rectangular region from the source buffer to the active drawing surface, copying from (srcLeft, srcTop, srcLeft+width-1, srcTop+height-1) in the source buffer to (dstLeft, dstTop) in the active buffer. The specified rop3 code is used to combine the pixels in the active buffer, along the currently active 8x8 color pattern.

### See Also

*BitBltBuf*, *BitBltPattBuf*, *SrcTransBltBuf*, *DstTransBltBuf*, *BitBltPlaneMaskedBuf*, *BitBltFxBuf*, *DrawRectBuf*

## BitBltFxBuf

Copies pixels from one buffer into the currently active buffer while stretching or shrinking to fit the destination

### Declaration

```
void GA_bufferFuncs::BitBltFxBuf(
    GA_buf *buf,
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 srcWidth,
    N_int32 srcHeight,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 dstWidth,
    N_int32 dstHeight,
    GA_bltFx *fx)
```

### Prototype In

snap/graphics.h

### Parameters

<i>buf</i>	SNAP buffer to blit to the active surface
<i>srcLeft</i>	Left coordinate of the source rectangle to copy
<i>srcTop</i>	Top coordinate of the source rectangle to copy
<i>srcWidth</i>	Width of the source rectangle in pixels
<i>srcHeight</i>	Height of the source rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>dstWidth</i>	Width of the destination rectangle in pixels
<i>dstHeight</i>	Height of the destination rectangle in scanlines
<i>fx</i>	<i>GA_bltFx</i> structure describing the requested effects

### Description

This function copies a rectangular region from the source buffer to the currently active buffer with the optional effects described in the *GA\_bltFx* structure. Currently this function can perform stretching, source and destination transparency and flipping depending on what features the underlying hardware supports, with an optional mix code. This routine will copy the rectangular region from (srcLeft, srcTop, srcLeft+srcWidth-1, srcTop+srcHeight-1) in the source buffer to (dstLeft, dstTop, dstLeft+dstWidth-1, dstTop+dstHeight-1) in the current active buffer. Note that the source and destination rectangle dimensions may be different in, which is the case for doing a copy with bitmap stretching. If the *GA\_bltFx* structure does not indicate stretching is in effect, the dstHeight and dstWidth parameters will be ignored and only the srcWidth and srcHeight parameters will be used.

---

**Note:** *Some of the features may not be supported at the same time, and it is up to the application programmer to call the BitBltFxTest function to determine what features are supported before calling this function. Calling this function with an unsupported set of features will result in undefined behaviour.*

---

**See Also**

*BitBltBuf, BitBltPattBuf, BitBltColorPattBuf, SrcTransBltBuf, DstTransBltBuf, DrawRectBuf, BitBltFxTest*

## BitBltPattBuf

Copies pixels from one buffer into the currently active buffer, applying a mono 8x8 pattern

### Declaration

```
void GA_bufferFuncs::BitBltPattBuf(
    GA_buf *buf,
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 width,
    N_int32 height,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 rop3)
```

### Prototype In

snap/graphics.h

### Parameters

<i>buf</i>	SNAP buffer to blit to the active surface
<i>srcLeft</i>	Left coordinate of the source rectangle to copy
<i>srcTop</i>	Top coordinate of the source rectangle to copy
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>rop3</i>	ROP3 code for the copy ( <i>GA_rop3CodesType</i> )

### Description

This function copies a rectangular region from the source buffer to the active drawing surface, copying from (srcLeft, srcTop, srcLeft+width-1, srcTop+height-1) in the source buffer to (dstLeft, dstTop) in the active buffer. The specified rop3 code is used to combine the pixels in the active buffer, along the currently active 8x8 monochrome pattern.

### See Also

*BitBltBuf*, *BitBltColorPattBuf*, *SrcTransBltBuf*, *DstTransBltBuf*, *BitBltPlaneMaskedBuf*, *BitBltFxBuf*, *DrawRectBuf*

## BitBltPlaneMaskedBuf

Copies pixels from one buffer into the currently active buffer while stretching or shrinking to fit the destination

### Declaration

```
void GA_bufferFuncs::BitBltPlaneMaskedBuf(
    GA_buf *buf,
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 width,
    N_int32 height,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_uint32 planeMask)
```

### Prototype In

snap/graphics.h

### Parameters

<i>buf</i>	SNAP buffer to blit to the active surface
<i>srcLeft</i>	Left coordinate of the source rectangle to copy
<i>srcTop</i>	Top coordinate of the source rectangle to copy
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>planeMask</i>	8-bit plane mask to use for the copy

### Description

This function copies a rectangular region from the source buffer to the currently active buffer. This routine will copy a rectangular region from (srcLeft, srcTop, srcLeft+width-1, srcTop+height-1) in the source buffer to (dstLeft, dstTop) in the currently active buffer, using the specified plane mask. The plane mask is used to determine which bits in the destination pixels will be affected by the copy. Each bit in the plane mask is used to mask out a bit in the destination pixel values, and where a bit is a 1 the destination bit comes from the source pixel while where a bit is 0 the destination bit is left unchanged.

### See Also

*BitBltBuf*, *BitBltPattBuf*, *BitBltColorPattBuf*, *SrcTransBltBuf*, *DstTransBltBuf*, *BitBltExBuf*, *DrawRectBuf*

## DrawRectBuf

Draws a solid filled rectangle with specific color and mix

### Declaration

```
void GA_bufferFuncs::DrawRectBuf (
    GA_buf *buf,
    N_int32 left,
    N_int32 top,
    N_int32 width,
    N_int32 height,
    GA_color color,
    N_int32 mix)
```

### Prototype In

snap/graphics.h

### Parameters

<i>buf</i>	SNAP buffer to draw to
<i>left</i>	Left coordinate of the rectangle to draw
<i>top</i>	Top coordinate of the rectangle to draw
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>color</i>	Color to fill the rectangle with
<i>mix</i>	Mix to fill the rectangle with

### Description

This function is used to draw a rectangle in the specified color and mix to a specific buffer. This function is useful when a buffer needs to be cleared to a solid color, but it is not the currently active buffer. Using this function avoids the overhead of switching the currently active buffer and then using the regular *DrawRect* functions.

### See Also

*BitBlkBuf*

## DstTransBltBuf

Copies pixels from one buffer into the currently active buffer, with destination color key transparency

### Declaration

```
void GA_bufferFuncs::DstTransBltBuf(
    GA_buf *buf,
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 width,
    N_int32 height,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 mix,
    GA_color transparent)
```

### Prototype In

snap/graphics.h

### Parameters

<i>buf</i>	SNAP buffer to blit to the active surface
<i>srcLeft</i>	Left coordinate of the source rectangle to copy
<i>srcTop</i>	Top coordinate of the source rectangle to copy
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>mix</i>	Mix code for the copy ( <i>GA_mixCodesType</i> )
<i>transparent</i>	Transparent color for the blit

### Description

This function copies a rectangular region from the source buffer to the active drawing surface, copying from (srcLeft, srcTop, srcLeft+width-1, srcTop+height-1) in the source buffer to (dstLeft, dstTop) in the active buffer. The specified mix is used to combine the pixels in the active buffer along with destination color key transparency.

The transparent color passed will be used to *mask out* pixels in the destination bitmap from being written. Where a pixel in the destination bitmap matches the transparent color, the pixel will be written to the destination bitmap.

---

**Note:** *Although you can achieve the same effect of this routine using the generic BitBlBufFx function, this function is provided separately as it is usually a workhorse function for sprite based game applications and needs to be as efficient as possible.*

---

### See Also

*BitBlBuf, BitBltPattBuf, BitBltColorPattBuf, SrcTransBltBuf, BitBltPlaneMaskedBuf, BitBltFxBuf, DrawRectBuf*



## FlipToBuffer

Makes a flippable buffer visible to the user

### Declaration

```
void GA_bufferFuncs::FlipToBuffer(
    GA_buf *buf,
    N_int32 waitVRT)
```

### Prototype In

snap/graphics.h

### Parameters

<i>buf</i>	SNAP buffer to make visible for display
<i>waitVRT</i>	Flags how to wait for vertical retraces ( <i>GA_MakeVisibleBufferFlagsType</i> )

### Description

This function makes the passed in buffer visible for the active display. The buffer does not become visible immediately, but will become visible on the next vertical retrace. The waitVRT flag however determines how the function will wait for the vertical retrace when programming flipping to the visible buffer. The values you can pass in are defined in the *GA\_MakeVisibleBufferFlagsType* enumeration.

If you call this function with waitVRT set to *gaTripleBuffer*, you can later call the *GetFlipStatus* function to determine if the visible buffer flip has occurred yet or not.

### See Also

*SetActiveBuffer*, *FlipToStereoBuffer*, *GetFlipStatus*, *WaitTillFlipped*

## FlipToStereoBuffer

Makes a stereo flippable buffer pair visible to the user

### Declaration

```
void GA_bufferFuncs::FlipToStereoBuffer(
    GA_buf *left,
    GA_buf *right,
    N_int32 waitVRT)
```

### Prototype In

snap/graphics.h

### Parameters

<i>left</i>	SNAP buffer to make visible for the left eye
<i>right</i>	SNAP buffer to make visible for the right eye
<i>waitVRT</i>	Flags how to wait for vertical retraces ( <i>GA_MakeVisibleBufferFlagsType</i> )

### Description

This function makes the passed in stereo buffers visible for the active display in stereo mode. If stereo mode is enabled, the left and right buffers are enabled for stereo flipping, alternating between the left and right buffers every vertical retrace period. If stereo is not enabled, the visible buffer becomes the left buffer and the right buffer value is ignored. The waitVRT flag however determines how the function will wait for the vertical retrace when programming flipping to the visible buffer. The values you can pass in are defined in the *GA\_MakeVisibleBufferFlagsType* enumeration.

If you call this function with waitVRT set to *gaTripleBuffer*, you can later call the *GetFlipStatus* function to determine if the visible buffer flip has occurred yet or not.

### See Also

*SetActiveBuffer*, *FlipToBuffer*, *GetFlipStatus*, *WaitTillFlipped*

## FreeBuffer

Destroys a buffer and frees the memory associated with it

### Declaration

```
ibool GA_bufferFuncs::FreeBuffer(  
    GA_buf *buf)
```

### Prototype In

snap/graphics.h

### Parameters

*buf*                      Buffer to destroy

### Return Value

True on success, false on failure.

### Description

This function destroys a SNAP buffer, and frees all resources associated with the buffer.

SEE ALSO *AllocBuffer*

## GetClipper

Returns the currently active clipper for a buffer

### Declaration

```
GA_clipper * GA_bufferFuncs::GetClipper(  
    GA_buf *buf)
```

### Prototype In

snap/graphics.h

### Parameters

*buf*                SNAP buffer to draw to

### Return Value

Pointer to currently active clipper, NULL if none assigned.

### Description

This function is used obtain a pointer to the currently active clipper object for the buffer. If no clipper object has been attached to the buffer, this function will return NULL (by default all buffer objects are created with no clipper objects attached to them).

### See Also

*BitBltnBuf, GetClipper*

## GetFlipStatus

Returns the status of the last scheduled buffer flip.

### Declaration

```
int GA_bufferFuncs::GetFlipStatus(void)
```

### Prototype In

snap/graphics.h

### Return Value

0 if flip has not occurred, not 0 if it has

### Description

This function updates returns the flip status for the currently pending flip operation. 1 means it has been flipped, 0 means it has not yet been flipped.

### See Also

*SetActiveBuffer, FlipToBuffer, FlipToStereoBuffer, WaitTillFlipped*

## GetFlippableBuffer

Returns a pointer to the requested flippable display buffer

### Declaration

```
GA_buf * GA_bufferFuncs::GetFlippableBuffer(  
    N_int32 index)
```

### Prototype In

snap/graphics.h

### Parameters

*index*                      Index of the flippable buffer to obtain the address of

### Return Value

Pointer to specified flippable buffer, or NULL if not such buffer.

### Description

This function returns a pointer to the flippable buffer with the specified index in the flippable buffer table. These buffer pointers can then be used with the *SetActiveBuffer* and *FlipToBuffer* functions to implement hardware page flipping.

### See Also

*AllocBuffer, GetPrimaryBuffer, SetActiveBuffer, FlipToBuffer*

## GetPrimaryBuffer

Returns a pointer to the primary buffer display buffer

### Declaration

```
GA_buf * GA_bufferFuncs::GetPrimaryBuffer(void)
```

### Prototype In

snap/graphics.h

### Return Value

Pointer to the primary display buffer, NULL if no primary buffer.

### Description

This function returns a pointer to the primary display buffer, which is always the first buffer on the fixed heap for regular display output.

### See Also

*AllocBuffer, FlipToBuffer, SetActiveBuffer, GetFlippableBuffer*

## InitBuffers

Initialises the buffer manager functions

### Declaration

```
ibool GA_bufferFuncs::InitBuffers(
    N_int32 numBuffers,
    N_uint32 flags,
    GA_softStereoFuncs *stereo)
```

### Prototype In

snap/graphics.h

### Parameters

<i>numBuffers</i>	Number of flippable buffers to allocate
<i>flags</i>	Extra flags for allocating buffers ( <i>GA_BufferFlagsType</i> )
<i>stereo</i>	Software stereo functions, or NULL for non-stereo mode

### Return Value

True on success, false on failure.

### Description

This function initialises the buffer manager, and allocates the primary buffer and all related flippable buffers (for a total of numBuffers buffers). If any of the buffers cannot be allocated, this function will return false. If the stereo parameter is not NULL, stereo mode is enabled using the passed in software stereo functions.

### See Also

*AllocBuffer, GetPrimaryBuffer, FlipToBuffer, SetActiveBuffer, BitBlitBuf*



## LockBuffer

Locks a buffer for direct memory access

### Declaration

```
N_uint32 GA_bufferFuncs::LockBuffer(
    GA_buf *buf)
```

### Prototype In

snap/graphics.h

### Parameters

*buf*                      Buffer to lock

### Return Value

Physical address of locked buffer in memory, 0 if a system memory buffer.

### Description

This function locks a buffer so that an application can begin drawing directly on the surface memory. You *must* call this function before you draw directly on the bitmap surface! After this function is called, the *GA\_bufSurface* member will be set to the virtual address of the buffer. Prior to calling this function and after calling *UnlockBuffer*, the *Surface* member will be set to NULL (indicating you should not access the memory!).

The return value from this function is the physical start address of the buffer in memory, which can be used to program DMA operations from other hardware devices directly into the video memory for the buffer. This is useful for frame grabber devices so that the resulting frame from the frame grabber device can be blitted to the visual display as quickly as possible (much quicker than if it was DMA'ed into system memory).

If the buffer is a system memory buffer, the return value from this function will be 0, since we cannot obtain the physical starting address of a system memory buffer.

SEE ALSO *UnlockBuffer*

## SetActiveBuffer

Makes a buffer the active buffer for all rendering functions

### Declaration

```
N_int32 GA_bufferFuncs::SetActiveBuffer(  
    GA_buf *buf)
```

### Prototype In

snap/graphics.h

### Parameters

*buf*                      SNAP buffer to make active for rendering

### Return Value

True on success, false on failure.

### Description

This function makes the passed in buffer the active buffer for all subsequent rendering operations. You *must* call this function in order to enable hardware rendering to the buffer. Once this function is called, all hardware rendering functions in the *GA\_2DRenderFuncs* structure will go to the newly active buffer.

SEE ALSO *FlipToBuffer*, *GetPrimaryBuffer*, *GetFlipBuffer*, *AllocBuffer*

## SetClipper

Set the currently active clipper for a buffer

### Declaration

```
void GA_bufferFuncs::SetClipper(  
    GA_buf *buf,  
    GA_clipper *clipper)
```

### Prototype In

snap/graphics.h

### Parameters

<i>buf</i>	SNAP buffer to draw to
<i>clipper</i>	New clipper to make the active clipper for the buffer

### Return Value

Pointer to currently active clipper, NULL if none assigned.

### Description

This function is used set the currently active clipper object for a buffer. Once a clipper object has been attached to a buffer, all rendering functions will be clipped to the dimensions of the attached clipper object. This functionality is only presently supported by the DirectX SNAP emulation driver, specifically to allow rendering directly in a window on the desktop.

### See Also

*BitBltBuf*, *GetClipper*

## SrcTransBltBuf

Copies pixels from one buffer into the currently active buffer, with source color key transparency

### Declaration

```
void GA_bufferFuncs::SrcTransBltBuf(
    GA_buf *buf,
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 width,
    N_int32 height,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 mix,
    GA_color transparent)
```

### Prototype In

snap/graphics.h

### Parameters

<i>buf</i>	SNAP buffer to blit to the active surface
<i>srcLeft</i>	Left coordinate of the source rectangle to copy
<i>srcTop</i>	Top coordinate of the source rectangle to copy
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>mix</i>	Mix code for the copy ( <i>GA_mixCodesType</i> )
<i>transparent</i>	Transparent color for the blit

### Description

This function copies a rectangular region from the source buffer to the active drawing surface, copying from (srcLeft, srcTop, srcLeft+width-1, srcTop+height-1) in the source buffer to (dstLeft, dstTop) in the active buffer. The specified mix is used to combine the pixels in the active buffer along with source color key transparency.

The transparent color passed will be used to *mask out* pixels in the source bitmap from being written to the destination area. Where a pixel in the source bitmap matches the transparent color, the pixel will not be written to the destination bitmap.

---

**Note:** *Although you can achieve the same effect of this routine using the generic BitBlBufFx function, this function is provided separately as it is usually a workhorse function for sprite based game applications and needs to be as efficient as possible.*

---

### See Also

*BitBlBuf, BitBltPattBuf, BitBltColorPattBuf, DstTransBltBuf, BitBltPlaneMaskedBuf, BitBltFxBuf, DrawRectBuf*

## StretchBltBuf

Copies pixels from one buffer into the currently active buffer while stretching or shrinking to fit the destination

### Declaration

```
void GA_bufferFuncs::StretchBltBuf(
    GA_buf *buf,
    N_int32 srcLeft,
    N_int32 srcTop,
    N_int32 srcWidth,
    N_int32 srcHeight,
    N_int32 dstLeft,
    N_int32 dstTop,
    N_int32 dstWidth,
    N_int32 dstHeight,
    N_int32 doClip,
    N_int32 clipLeft,
    N_int32 clipTop,
    N_int32 clipRight,
    N_int32 clipBottom,
    N_int32 mix)
```

### Prototype In

snap/graphics.h

### Parameters

<i>buf</i>	SNAP buffer to blit to the active surface
<i>srcLeft</i>	Left coordinate of the source rectangle to copy
<i>srcTop</i>	Top coordinate of the source rectangle to copy
<i>srcWidth</i>	Width of the source rectangle in pixels
<i>srcHeight</i>	Height of the source rectangle in scanlines
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>dstWidth</i>	Width of the destination rectangle in pixels
<i>dstHeight</i>	Height of the destination rectangle in scanlines
<i>doClip</i>	True if the blit should be clipped
<i>clipLeft</i>	Left coordinate of clip rectangle
<i>clipTop</i>	Top coordinate of clip rectangle
<i>clipRight</i>	Right coordinate of clip rectangle
<i>clipBottom</i>	Bottom coordinate of clip rectangle
<i>mix</i>	Mix code for the copy ( <i>GA_mixCodesType</i> )

### Description

This function copies a rectangular region of source buffer to the currently active buffer with either stretching or shrinking. This routine will copy the rectangular region of from (srcLeft, srcTop, srcLeft+srcWidth-1, srcTop+srcHeight-1) in the source buffer to (dstLeft, dstTop, dstLeft+dstWidth-1, dstTop+dstHeight-1) in the active buffer. Note that the source and destination rectangle dimensions may be different in, which is the case for doing a copy with bitmap stretching or shrinking.

If the doClip parameter is true, then the output of the stretch function will be clipped against the passed in destination clip rectangle.

**See Also**

*BitBltBuf, BitBltPattBuf, BitBltColorPattBuf, SrcTransBltBuf, DstTransBltBuf, BitBltPlaneMaskedBuf, BitBltFxBuf, DrawRectBuf*

## UnlockBuffer

Unlocks a buffer after direct memory access

### Declaration

```
void GA_bufferFuncs::UnlockBuffer(  
    GA_buf *buf)
```

### Prototype In

snap/graphics.h

### Parameters

*buf*                  SNAP buffer to unlock

### Description

This function unlocks a buffer after the application has completed direct surface access on the buffer.

SEE ALSO *LockBuffer*

## UpdateCache

Updates the system cache from the video memory buffer contents

### Declaration

```
void GA_bufferFuncs::UpdateCache(  
    GA_buf *buf)
```

### Prototype In

snap/graphics.h

### Parameters

*buf*                SNAP buffer to copy into system memory cache

### Description

This function is used to update the system memory buffer cache by copying the contents of the video memory buffer into the system memory cache. This is mostly useful if you are using cached buffers with a system memory shadow, and you have updated the video memory buffer and need to flush the changes to the system memory cache. This operation is not particularly fast (video memory reads are always slow), but if you need to keep the system memory cache up to date this is the way to do it.

### See Also

*UpdateFromCache, BitBlitBuf*



## UpdateFromCache

Updates the video memory buffer contents from system memory cache

### Declaration

```
void GA_bufferFuncs::UpdateFromCache(  
    GA_buf *buf)
```

### Prototype In

snap/graphics.h

### Parameters

*buf*                SNAP buffer to copy from system memory cache

### Description

This function updates the video memory buffer by copying the contents of the system memory cache buffer into the video memory buffer. This is useful if you need to replace the buffer contents with new values, or you need to do software rendering on the buffer. Doing the rendering on the system memory buffer will be faster, and when you are done this function can be used to update the video memory copy of the buffer. Unless you need to specifically do some drawing in hardware, updating the system memory cache and using this function will be faster than updating video memory and using *UpdateCache*.

### See Also

*UpdateCache*, *BitBltBuf*

## WaitTillFlipped

Waits until the last scheduled buffer flip occurs

### Declaration

```
void GA_bufferFuncs::WaitTillFlipped(void)
```

### Prototype In

snap/graphics.h

### Description

This function waits until the pending flip operation has completed and the visible display page has changed before returning.

### See Also

*SetActiveBuffer, FlipToBuffer, FlipToStereoBuffer, GetFlipStatus*

## GA\_busType

### Declaration

```
typedef enum {
    gaUnknownBus          = 0,
    gaISABus              = 1,
    gaMCABus              = 2,
    gaVLBBus              = 3,
    gaPCIBus              = 4,
    gaAGPBus              = 5
} GA_busType
```

### Prototype In

snap/graphics.h

### Description

This enumeration defines the values stored in the BusType field of the *GA\_devCtx* structure.

### Members

<i>gaUnknownBus</i>	Bus type is not known
<i>gaISABus</i>	Device is an ISA bus device
<i>gaMCABus</i>	Device is a Micro-Channel bus device
<i>gaVLBBus</i>	Device is a VESA Local Bus device
<i>gaPCIBus</i>	Device is a PCI bus device
<i>gaAGPBus</i>	Device is an AGP bus device

## GA\_certifyChipInfo

### Declaration

```
typedef struct {
    N_uint32    dwSize;
    char        ChipsetName[30];
    N_uint16    CertifyVersion;
    char        CertifiedDate[19];
    N_uint8     CertifyFlags;
} GA_certifyChipInfo
```

### Prototype In

snap/graphics.h

### Description

Structure pointed to be the *GA\_certifyInfo* structure, which contains certification information about the specific chipsets in the device driver.

---

**Note:** *The dwSize member is intended for future compatibility, and should be set to the size of the structure as defined in this header file. Future drivers will be compatible with older software by examining this value.*

---

### Members

<i>dwSize</i>	Set to size of structure in bytes
<i>ChipsetName</i>	Name of graphics chipset name
<i>CertifyVersion</i>	Version of certification program used
<i>CertifiedDate</i>	Date that the card was certified
<i>CertifyFlags</i>	Flags for certification information

## GA\_certifyInfo

### Declaration

```
typedef struct {
    N_uint32          dwSize;
    char              Signature[20];
    char              BuildDate[30];
    char              MaxCertifiedChips;
    char              NumCertifiedChips;
    GA_certifyChipInfo *CertifiedCards;
} GA_certifyInfo
```

### Prototype In

snap/graphics.h

### Description

Structure returned by GetCertifyInfo, which contains configuration information about the certification status of the drivers.

---

**Note:** *The dwSize member is intended for future compatibility, and should be set to the size of the structure as defined in this header file. Future drivers will be compatible with older software by examining this value.*

---

### Members

<i>dwSize</i>	Set to size of structure in bytes
<i>Signature</i>	Signature to identify certification information
<i>BuildDate</i>	String representation of the build date for driver
<i>MaxCertifiedCards</i>	Maximum number of certified chipsets in driver
<i>NumCertifiedCards</i>	Number of certified chipsets
<i>CertifiedCards</i>	List of all certified cards in the driver

## GA\_clipper

**Declaration**

```
typedef void GA_clipper
```

**Prototype In**

snap/graphics.h

**Description**

Defines the fundamental type for a SNAP Graphics clipper object. The internals of this object are completely implementation dependant so we simply define this type as a void pointer as the application code should never care about the internals of a *GA\_clipper* object.

## GA\_clipperFuncs

### Prototype In

snap/graphics.h

### Description

Function group containing all offscreen buffer management window clipper functions available via the SNAP API's. These functions manage the creation and destruction of complex clip regions for handling clipping to window manager windows. These functions manage translation of OS specific clip regions to SciTech SNAP Graphics complex clip regions. This function group is *only* returned by the 2D reference rasteriser library, and not by hardware drivers.

---

**Note:** *Be sure to fill in the dwSize member of this structure when you call GA\_queryFunctions to the correct size of the structure at compile time!*

---

## CreateClipper

Creates a clipper object for a specific window manager window

### Declaration

```
GA_clipper * GA_clipperFuncs::CreateClipper(  
    PM_HWND hwnd)
```

### Prototype In

snap/graphics.h

### Parameters

*hwnd*                      Window handle to create the clipper for

### Return Value

Pointer to the allocated clipper, NULL on failure.

### Description

This function is used to create a clipper object that is associated with a window manager window handle. This clipper object will track the visible clip lists associated with the window handle, and when attached to a buffer will cause all output to that buffer to be clipped to the currently active visible clip list for the window.

This function is presently only implemented in the SNAP DirectX emulation driver, and is intended to allow the SNAP to draw directly into windows on the desktop.

### See Also

*IsClipListChanged, GetClipList, DestroyClipper*



## DestroyClipper

Destroys a clipper object

### Declaration

```
void GA_clipperFuncs::DestroyClipper(  
    GA_clipper *clipper)
```

### Prototype In

snap/graphics.h

### Parameters

*clipper*                      Clipper object to destroy

### Description

This function destroys a previously allocated clipper object, freeing all memory associated with the clipper object.

This function is presently only implemented in the SNAP DirectX emulation driver, and is intended to allow the SNAP to draw directly into windows on the desktop.

### See Also

*CreateClipper, IsClipListChanged, GetClipList*

## GetClipList

Returns a list of rectangles representing the complex clip list

### Declaration

```
GA_rect * GA_clipperFuncs::GetClipList(
    GA_clipper *clipper,
    N_int32 *count)
```

### Prototype In

snap/graphics.h

### Parameters

<i>clipper</i>	Clipper object to test
<i>count</i>	Number of rectangles in clip list

### Return Value

Pointer to a list of rectangles in the clip list.

### Description

This function returns the complex clip list for the associated clipper if the window requires complex clipping. If the region does not require complex clipping, this function will return a single rectangle and return a value of 1 in the count parameter. The complex clip list is just a list of rectangles, and the number of rectangles in the list is returned in the count parameter.

This function is presently only implemented in the SNAP DirectX emulation driver, and is intended to allow the SNAP to draw directly into windows on the desktop.

### See Also

*CreateClipper, IsClipListChanged, DestroyClipper*

## IsClipListChanged

Determines if the window clip list has changed

### Declaration

```
ibool GA_clipperFuncs::IsClipListChanged(  
    GA_clipper *clipper)
```

### Prototype In

snap/graphics.h

### Parameters

*clipper*                      Clipper object to test

### Return Value

True if the window clip list has changed, false if not.

### Description

This function is used to determine if the window clip list has changed for a clipper object since the last time this function was called.

This function is presently only implemented in the SNAP DirectX emulation driver, and is intended to allow the SNAP to draw directly into windows on the desktop.

---

**Note:** *This function always returns true the first time it is called.*

---

### See Also

*CreateClipper, GetClipList, DestroyClipper*

## GA\_color

### Declaration

```
typedef N_uint32 GA_color
```

### Prototype In

snap/graphics.h

### Description

Fundamental type definition for a 32-bit color value. The color value is interpreted differently depending on what graphics mode the system is in, and in 15-bit and above modes will have the color values packed according to the pixel format information stored in the *GA\_modeInfo* structure.

## GA\_colorCursor

### Declaration

```
typedef struct {
    N_uint8    ColorData[2048];
    N_uint8    ANDMask[512];
    GA_palette  Palette[16];
    N_uint32    HotX;
    N_uint32    HotY;
} GA_colorCursor
```

### Prototype In

snap/graphics.h

### Description

Hardware 16-color cursor structure. This structure defines a color hardware cursor that is downloaded to the hardware. The cursor is defined as a 64x64 image with an AND mask and color data. The definition of the AND mask, cursor data and the pixels that will appear on the screen is as follows:

AND	Color	Result
0	0	Transparent (color from screen memory)
0	not 0	Invert (complement of color from screen memory)
1	xx	Cursor color data

Hence if the AND mask is a zero the color data should be either 00 to make the pixel transparent or not 0 to make it the inversion of the screen pixel.

The color data is passed down to the driver as 4-bit packed color index values, along with a 16-color lookup table containing the real 24-bit RGB color values for the cursor image. It is up to the calling application to translate and quantise cursor images of higher color depths down to the format supported by the hardware.

The HotX and HotY members define the *hot spot* for the cursor, which is the location where the logical mouse pointer is located in the cursor image. When you click the mouse, the pixel under the hot-spot is the pixel selected.

### Members

<i>ColorData</i>	Cursor color data as a 64x64 array of packed 4-bit pixels
<i>ANDMask</i>	Cursor AND mask
<i>Palette</i>	16-color palette for cursor image
<i>HotX</i>	Cursor X coordinate hot spot
<i>HotY</i>	Cursor Y coordinate hot spot

## GA\_colorCursor256

### Declaration

```
typedef struct {
    N_uint8    ColorData[4096];
    N_uint8    ANDMask[512];
    GA_palette  Palette[256];
    N_uint32    HotX;
    N_uint32    HotY;
} GA_colorCursor256
```

### Prototype In

snap/graphics.h

### Description

Hardware 256-color cursor structure. This structure defines a color hardware cursor that is downloaded to the hardware. The cursor is defined as a 64x64 image with an AND mask and color data. The definition of the AND mask, cursor data and the pixels that will appear on the screen is as follows:

AND	Color	Result
0	0	Transparent (color from screen memory)
0	not 0	Invert (complement of color from screen memory)
1	xx	Cursor color data

Hence if the AND mask is a zero the color data should be either 00 to make the pixel transparent or not 0 to make it the inversion of the screen pixel.

The color data is passed down to the driver as 8-bit packed color index values, along with a 256-color lookup table containing the real 24-bit RGB color values for the cursor image. It is up to the calling application to translate and quantise cursor images of higher color depths down to the format supported by the hardware.

The HotX and HotY members define the *hot spot* for the cursor, which is the location where the logical mouse pointer is located in the cursor image. When you click the mouse, the pixel under the hot-spot is the pixel selected.

### Members

<i>ColorData</i>	Cursor color data as a 64x64 array of packed 8-bit pixels
<i>ANDMask</i>	Cursor AND mask
<i>Palette</i>	256-color palette for cursor image
<i>HotX</i>	Cursor X coordinate hot spot
<i>HotY</i>	Cursor Y coordinate hot spot

## GA\_colorCursorRGB

### Declaration

```
typedef struct {
    N_uint8    ColorData[12288];
    N_uint8    ANDMask[512];
    N_uint32    HotX;
    N_uint32    HotY;
} GA_colorCursorRGB
```

### Prototype In

snap/graphics.h

### Description

Hardware 24-bit cursor structure. This structure defines a color hardware cursor that is downloaded to the hardware. The cursor is defined as a 64x64 image with an AND mask and color data. The definition of the AND mask, cursor data and the pixels that will appear on the screen is as follows:

AND	Color	Result
0	0	Transparent (color from screen memory)
0	not 0	Invert (complement of color from screen memory)
1	xx	Cursor color data

Hence if the AND mask is a zero the color data should be either 00 to make the pixel transparent or not 0 to make it the inversion of the screen pixel.

The color data is passed down to the driver as 24-bit packed RGB color values. It is up to the calling application to translate cursor images of lower color depths to the format supported by the hardware.

The HotX and HotY members define the *hot spot* for the cursor, which is the location where the logical mouse pointer is located in the cursor image. When you click the mouse, the pixel under the hot-spot is the pixel selected.

### Members

<b>ColorData</b>	Cursor color data as a 64x64 array of packed 24-bit RGB pixels
<b>ANDMask</b>	Cursor AND mask
<b>HotX</b>	Cursor X coordinate hot spot
<b>HotY</b>	Cursor Y coordinate hot spot

## GA\_colorCursorRGBA

### Declaration

```
typedef struct {
    N_uint8      ColorData[16384];
    N_uint32     HotX;
    N_uint32     HotY;
} GA_colorCursorRGBA
```

### Prototype In

snap/graphics.h

### Description

Hardware 24-bit RGBA alpha blended cursor structure. This structure defines a color hardware cursor that is downloaded to the hardware. The cursor is defined as a 64x64 24-bit RGBA image with alpha channel. The alpha channel data is used to define the transparency level for the bitmap, with 0 being fully transparent and 255 being full opaque. Since the color bitmap data is alpha blended, there is no AND mask for the cursor image.

The HotX and HotY members define the *hot spot* for the cursor, which is the location where the logical mouse pointer is located in the cursor image. When you click the mouse, the pixel under the hot-spot is the pixel selected.

### Members

<i>ColorData</i>	Cursor color data as a 64x64 array of packed 24-bit RGBA pixels
<i>HotX</i>	Cursor X coordinate hot spot
<i>HotY</i>	Cursor Y coordinate hot spot



## GA\_colorPattern

### Declaration

```
typedef union {  
    GA_colorPattern_1    b1;  
    GA_colorPattern_4    b4;  
    GA_colorPattern_8    b8;  
    GA_colorPattern_16   b16;  
    GA_colorPattern_24   b24;  
    GA_colorPattern_32   b32;  
} GA_colorPattern
```

### Prototype In

snap/graphics.h

### Description

Fundamental type definition for an array element of an 8x8 color bitmap pattern. Each line in the pattern is represented as an array of packed pixel data. In 8bpp modes there is 8 bytes per line, for 16bpp modes there are 16bytes per line, for 24bpp modes there are 24bytes per line and for 32bpp modes there are 32 bytes per line. Hence the size of the pattern data is different depending on the color depth currently active.

## GA\_colorPattern\_1

### Declaration

```
typedef struct {  
    N_uint8      p[8];  
} GA_colorPattern_1
```

### Prototype In

snap/graphics.h

### Description

Fundamental type definition for an array element of an 8x8 color bitmap pattern data for 1bpp modes.

### Members

*p*            8x8 bytes of pattern data

## GA\_colorPattern\_16

### Declaration

```
typedef struct {  
    N_uint16    p[8][8];  
} GA_colorPattern_16
```

### Prototype In

snap/graphics.h

### Description

Fundamental type definition for an array element of an 8x8 color bitmap pattern data for 16bpp modes.

### Members

*p*            8x8 words of pattern data

## GA\_colorPattern\_24

### Declaration

```
typedef struct {  
    N_uint8      p[8][8][3];  
} GA_colorPattern_24
```

### Prototype In

snap/graphics.h

### Description

Fundamental type definition for an array element of an 8x8 color bitmap pattern data for 24bpp modes.

### Members

*p*            8x8x3 bytes of pattern data

## GA\_colorPattern\_32

### Declaration

```
typedef struct {  
    N_uint32    p[8][8];  
} GA_colorPattern_32
```

### Prototype In

snap/graphics.h

### Description

Fundamental type definition for an array element of an 8x8 color bitmap pattern data for 32pp modes.

### Members

*p*            8x8 dwords of pattern data

## GA\_colorPattern\_4

### Declaration

```
typedef struct {  
    N_uint8      p[8][4];  
} GA_colorPattern_4
```

### Prototype In

snap/graphics.h

### Description

Fundamental type definition for an array element of an 8x8 color bitmap pattern data for 4bpp modes.

### Members

*p*            8x8 bytes of pattern data

## GA\_colorPattern\_8

### Declaration

```
typedef struct {  
    N_uint8      p[8][8];  
} GA_colorPattern_8
```

### Prototype In

snap/graphics.h

### Description

Fundamental type definition for an array element of an 8x8 color bitmap pattern data for 8bpp modes.

### Members

*p*            8x8 bytes of pattern data

## GA\_configInfo

### Declaration

```
typedef struct {
    N_uint32      dwSize;
    char          ManufacturerName[80];
    char          ChipsetName[80];
    char          DACName[80];
    char          ClockName[80];
    char          VersionInfo[80];
    char          BuildDate[80];
    char          Certified;
    char          CertifiedDate[20];
    N_uint16      CertifyVersion;
} GA_configInfo
```

### Prototype In

snap/graphics.h

### Description

Structure returned by GetConfigInfo, which contains configuration information about the installed graphics hardware.

---

**Note:** *The dwSize member is intended for future compatibility, and should be set to the size of the structure as defined in this header file. Future drivers will be compatible with older software by examining this value.*

---

### Members

<i>dwSize</i>	Set to size of structure in bytes
<i>ManufacturerName</i>	Name of graphics chipset manufacturer
<i>ChipsetName</i>	Name of graphics chipset name
<i>DACName</i>	Name of DAC on graphics card
<i>ClockName</i>	Name of clock on graphics card
<i>VersionInfo</i>	String representation of version and build for driver
<i>BuildDate</i>	String representation of the build date for driver
<i>Certified</i>	True if the installed device is certified
<i>CertifiedDate</i>	Date when the device was certified
<i>CertifyVersion</i>	Version of certification program used



## GA\_cursorFuncs

### Prototype In

snap/graphics.h

### Description

Function group containing all hardware cursor functions available for the device. These functions are used to manage the hardware cursor functions for the device, such as setting the cursor, setting the cursor position and changing the cursor colors.

Note also that this function group is also returned by the 2D reference rasteriser code to implement software cursor functions when there is no hardware cursor support in the hardware. Unless you have a specific need to directly manage the hardware cursor, you should always use the functions provided by the 2D reference rasteriser. The 2D reference rasteriser will handle any hardware specific limitations for you, defaulting back to a software cursor when necessary to ensure correct operation.

---

**Note:** *Be sure to fill in the dwSize member of this structure when you call GA\_queryFunctions to the correct size of the structure at compile time!*

---

## BeginAccess

Begin drawing access to the framebuffer to exclude the software cursor image

### Declaration

```
void NAPI_GA_cursorFuncs::BeginAccess (
    N_int32 left,
    N_int32 top,
    N_int32 right,
    N_int32 bottom)
```

### Prototype In

snap/graphics.h

### Parameters

<i>left</i>	Left coordinate of display affected
<i>top</i>	Top coordinate of display affected
<i>right</i>	Right coordinate of display affected (exclusive)
<i>bottom</i>	Bottom coordinate of display affected (exclusive)

### Description

This function must be used by the application when the 2d reference rasteriser is using a software based cursor image. The cursor images are stored in either system memory or offscreen memory buffers, and either a software or hardware blitter is used to draw the cursor images on the screen. Since the cursor images are physically drawn on the display screen, this function must be used to inform the 2d reference rasteriser that the application is about to draw to a particular location on the screen. The 2d reference rasteriser will then remove the cursor image from the screen temporarily while the drawing is taking place if the region being drawn to and the cursor image overlap.

This function is not used when the cursor is implemented entirely in hardware on the display device. It will also not be provided by hardware drivers, just by the 2d reference rasteriser library. Optimised applications and shell drivers should disable calls to this function when a hardware cursor is in use (see the *IsHardwareCursor* function to determine this at runtime).

---

**Note:** *This function must always be used for all drawing operations, regardless of whether the drawing is being done in software or hardware or directly by the application program writing to the framebuffer.*

---

### See Also

*SetMonoCursor, SetMonoCursorColor, SetColorCursor, SetCursorPos, ShowCursor, BeginAccess, EndAccess, SetColorCursor256, SetColorCursorRGBA, IsHardwareCursor, SetColorCursorRGB*

## EndAccess

End access to the framebuffer for drawing

### Declaration

```
void NAPI_GA_cursorFuncs::EndAccess(void)
```

### Prototype In

snap/graphics.h

### Description

This function must be used by the application when the 2d reference rasteriser is using a software based cursor image. This function informs the 2d reference rasteriser that drawing has completed, and that it may restore the cursor image back to the display screen at it's original location (if it needed to be hidden).

This function is not used when the cursor is implemented entirely in hardware on the display device. It will also not be provided by hardware drivers, just by the 2d reference rasteriser library.

### See Also

*SetMonoCursor, SetMonoCursorColor, SetColorCursor, SetCursorPos, ShowCursor, BeginAccess, EndAccess, SetColorCursor256, SetColorCursorRGBA, IsHardwareCursor, SetColorCursorRGB*

## IsHardwareCursor

Determines if the current cursor is a hardware cursor or software cursor

### Declaration

```
N_int32 NAPI GA_cursorFuncs::IsHardwareCursor(void)
```

### Prototype In

snap/graphics.h

### Description

This function is used to determine if the currently active cursor is implemented in software or hardware. This function should be called immediately after downloading a new cursor images via the 2d reference rasteriser library. If the device driver is able to implement the cursor in hardware, this function will return true. If the 2d reference rasteriser library has to fall back to software, this function will return false. If this function does return false, the application program *must* use the *BeginAccess* and *EndAccess* functions to ensure the software cursor images is correctly excluded from the screen for all drawing operations.

This function will not be provided by hardware drivers, only by the 2d reference rasteriser library.

### See Also

*SetMonoCursor*, *SetMonoCursorColor*, *SetColorCursor*, *SetCursorPos*, *ShowCursor*, *BeginAccess*, *EndAccess*, *SetColorCursor256*, *SetColorCursorRGBA*, *IsHardwareCursor*, *SetColorCursorRGB*

## SetColorCursor

Sets a new 64x64 16-color hardware cursor image.

### Declaration

```
void NAPI GA_cursorFuncs::SetColorCursor(
    GA_colorCursor *cursor)
```

### Prototype In

snap/graphics.h

### Parameters

*cursor*                      Pointer to an *GA\_colorCursor* structure for the cursor image

### Description

This function downloads the specified 16-color cursor definition from the application into the hardware cursor. The cursor data is passed by the application in the *GA\_colorCursor* format; please see the documentation for this structure for the format of the data passed to this function.

---

**Note:** *To pad the cursor definition to 64x64 for cursors that are smaller than 64x64 in size, simply fill in the remainder of the XORMask and ANDMask with zeroes.*

---

### See Also

*SetMonoCursor, SetMonoCursorColor, SetColorCursor, SetCursorPos, ShowCursor, BeginAccess, EndAccess, SetColorCursor256, SetColorCursorRGBA, IsHardwareCursor, SetColorCursorRGB*

## SetColorCursor256

Sets a new 64x64 256-color hardware cursor image.

### Declaration

```
void NAPI GA_cursorFuncs::SetColorCursor256(
    GA_colorCursor256 *cursor)
```

### Prototype In

snap/graphics.h

### Parameters

*cursor*                      Pointer to an *GA\_colorCursor256* structure for the cursor image

### Description

This function downloads the specified 256-color cursor definition from the application into the hardware cursor. The cursor data is passed by the application in the *GA\_colorCursor256* format; please see the documentation for this structure for the format of the data passed to this function.

---

**Note:** *To pad the cursor definition to 64x64 for cursors that are smaller than 64x64 in size, simply fill in the remainder of the XORMask and ANDMask with zeroes.*

---

### See Also

*SetMonoCursor, SetMonoCursorColor, SetColorCursor, SetCursorPos, ShowCursor, BeginAccess, EndAccess, SetColorCursor256, SetColorCursorRGBA, IsHardwareCursor, SetColorCursorRGB*

## SetColorCursorRGB

Sets a new 64x64 24-bit color RGB TrueColor hardware cursor image.

### Declaration

```
void NAPI GA_cursorFuncs::SetColorCursorRGB(
    GA_colorCursorRGB *cursor)
```

### Prototype In

snap/graphics.h

### Parameters

*cursor*                      Pointer to an *GA\_colorCursorRGB* structure for the cursor image

### Description

This function downloads the specified 24-bit RGB TrueColor cursor definition from the application into the hardware cursor. The cursor data is passed by the application in the *GA\_colorCursorRGB* format; please see the documentation for this structure for the format of the data passed to this function.

---

**Note:** *To pad the cursor definition to 64x64 for cursors that are smaller than 64x64 in size, simply fill in the remainder of the XORMask and ANDMask with zeroes.*

---

### See Also

*SetMonoCursor, SetMonoCursorColor, SetColorCursor, SetCursorPos, ShowCursor, BeginAccess, EndAccess, SetColorCursor256, SetColorCursorRGBA, IsHardwareCursor, SetColorCursorRGB*

## SetColorCursorRGBA

Sets a new 64x64 32-bit color RGBA alpha blended hardware cursor image.

### Declaration

```
void NAPI GA_cursorFuncs::SetColorCursorRGBA(
    GA_colorCursorRGBA *cursor)
```

### Prototype In

snap/graphics.h

### Parameters

<i>cursor</i>	Pointer to an <i>GA_colorCursorRGBA</i> structure for the cursor image
---------------	--

### Description

This function downloads the specified 32-bit RGBA alpha blended cursor definition from the application into the hardware cursor. The cursor data is passed by the application in the *GA\_colorCursorRGBA* format; please see the documentation for this structure for the format of the data passed to this function.

---

**Note:** *To pad the cursor definition to 64x64 for cursors that are smaller than 64x64 in size, simply fill in the remainder of the XORMask and ANDMask with zeroes.*

---

### See Also

*SetMonoCursor, SetMonoCursorColor, SetColorCursor, SetCursorPos, ShowCursor, BeginAccess, EndAccess, SetColorCursor256, SetColorCursorRGBA, IsHardwareCursor, SetColorCursorRGB*



## SetCursorPos

Sets the hardware cursor position.

### Declaration

```
N_int32 NAPI GA_cursorFuncs::SetCursorPos(  
    N_int32 x,  
    N_int32 y)
```

### Prototype In

snap/graphics.h

### Parameters

<i>x</i>	X coordinate of the cursor position
<i>y</i>	Y coordinate of the cursor position

### Return Value

Ignored. The return value is obsolete and should always be ignored.

### Description

This function sets the location of the hardware cursor in display coordinates. This function takes the X and Y coordinates of the new cursor location. This function will place the cursor so that the hotspot of the currently active cursor image is located at the specified (X,Y) location, and will correctly handle special cases where the cursor image needs to be located off the edges of the display screen (such as when X=0, Y=0 and HotX,HotY > 0).

### See Also

*SetMonoCursor, SetMonoCursorColor, SetColorCursor, SetCursorPos, ShowCursor, BeginAccess, EndAccess, SetColorCursor256, SetColorCursorRGBA, IsHardwareCursor, SetColorCursorRGB*

## SetMonoCursor

Sets a new 64x64 monochrome hardware cursor image.

### Declaration

```
void NAPI GA_cursorFuncs::SetMonoCursor(  
    GA_monoCursor *cursor)
```

### Prototype In

snap/graphics.h

### Parameters

*cursor*                      Pointer to an *GA\_monoCursor* structure for the cursor image

### Description

This function downloads the specified monochrome cursor definition from the application into the hardware cursor. The cursor data is passed by the application in the *GA\_monoCursor* structure.

---

**Note:** *To pad the cursor definition to 64x64 for cursors that are smaller than 64x64 in size, simply fill in the remainder of the XORMask and ANDMask with zeroes.*

---

### See Also

*SetMonoCursor, SetMonoCursorColor, SetColorCursor, SetCursorPos, ShowCursor, BeginAccess, EndAccess, SetColorCursor256, SetColorCursorRGBA, IsHardwareCursor, SetColorCursorRGB*

## SetMonoCursorColor

Sets the foreground and background color for the monochrome hardware cursor.

### Declaration

```
void NAPI GA_cursorFuncs::SetMonoCursorColor (
    GA_palette *foreground,
    GA_palette *background)
```

### Prototype In

snap/graphics.h

### Parameters

<i>foreground</i>	Foreground color for the cursor
<i>background</i>	Background color for the cursor

### Description

This function sets both the foreground and background colors for the hardware cursor, which are passed in as *GA\_palette* structures. In 8bpp display modes the color index of the cursor color is passed in the Red member of *GA\_palette* structure. The exception to this is for 8bpp display modes that have the *gaHave8bppRGBCursor* flag set in the *GA\_modeInfo* Attributes field. When this is set, the full RGB color of the cursor is passed in the red, green and blue members of the *GA\_palette* structure.

For 15-bpp and above RGB modes, the full RGB color of the cursor is always passed in the red, green and blue members of the *GA\_palette* structure.

### See Also

*SetMonoCursor*, *SetMonoCursorColor*, *SetColorCursor*, *SetCursorPos*, *ShowCursor*, *BeginAccess*, *EndAccess*, *SetColorCursor256*, *SetColorCursorRGBA*, *IsHardwareCursor*, *SetColorCursorRGB*

## ShowCursor

Shows or hides the hardware cursor image.

### Declaration

```
void NAPI_GA_cursorFuncs::ShowCursor(  
    N_int32 visible)
```

### Prototype In

snap/graphics.h

### Parameters

*visible*                      1 to show the cursor, 0 to hide the cursor

### Description

This function unconditionally either show or hides the current active hardware cursor image.

### See Also

*SetMonoCursor, SetMonoCursorColor, SetColorCursor, SetCursorPos, ShowCursor, BeginAccess, EndAccess, SetColorCursor256, SetColorCursorRGBA, IsHardwareCursor, SetColorCursorRGB*

## GA\_devCtx

### Declaration

```

struct GA_devCtx {
    char        Signature[20];
    N_uint32    Version;
    N_uint32    DriverRev;
    char        OemVendorName[80];
    char        OemCopyright[80];
    N_uint16    _FAR_ *AvailableModes;
    N_int32     DeviceIndex;
    N_uint32    TotalMemory;
    N_uint32    Attributes;
    N_uint32    WorkArounds;
    N_uint32    TextSize;
    N_uint32    TextBasePtr;
    N_uint32    BankSize;
    N_uint32    BankedBasePtr;
    N_uint32    LinearSize;
    N_uint32    LinearBasePtr;
    N_uint32    ZBufferSize;
    N_uint32    ZBufferBasePtr;
    N_uint32    TexBufferSize;
    N_uint32    TexBufferBasePtr;
    N_uint32    LockedMemSize;
    N_uint32    IOBase;
    N_uint32    MMIOBase[4];
    N_uint32    MMIOLen[4];
    void        _FAR_ *DriverStart;
    N_uint32    DriverSize;
    N_uint32    BusType;
    N_uint32    AttributesExt;
    N_uint32    res1[18];

    void        _FAR_ *IOMemMaps[4];
    void        _FAR_ *TextMem;
    void        _FAR_ *BankedMem;
    void        _FAR_ *LinearMem;
    void        _FAR_ *ZBufferMem;
    void        _FAR_ *TexBufferMem;
    void        _FAR_ *LockedMem;
    N_uint32    LockedMemPhys;
    void        _FAR_ *TextFont8x8;
    void        _FAR_ *TextFont8x14;
    void        _FAR_ *TextFont8x16;
    GA_palette  _FAR_ *VGAPal4;
    GA_palette  _FAR_ *VGAPal8;
    N_uint32    res3[18];

    struct GA_devCtx _FAR_ *ring0DC;
    void        _FAR_ *pMdl;

    GA_loaderFuncs loader;
}

```

### Prototype In

snap/graphics.h

### Description

Main graphics device context structure. This structure consists of a header block that contains configuration information about the graphic device, as well as detection information and runtime state information.

The Signature member is filled with the null terminated string 'GRAPHICS\0' by the driver implementation. This can be used to verify that the file loaded really is an graphics device driver.

The Version member is a BCD value which specifies what revision level of the graphics specification is implemented in the driver. The high byte specifies the major version number and the low byte specifies the minor version number. For example, the BCD value for version 1.0 is 0x100 and the BCD value for version 2.2 would be 0x202.

The DriverRev member specifies the driver revision level, and is used by the driver configuration software to determine which version was used to generate the driver file.

The OemVendorName member contains the name of the vendor that developed the device driver implementation, and can be up to 80 characters in length.

The OemCopyright member contains a copyright string for the vendor that developed the device driver implementation and may be up to 80 characters in length.

The AvailableModes is an pointer within the loaded driver to a list of mode numbers for all displaymodes supported by the graphics driver. Each mode number occupies one word (16-bits), and is terminated by a -1 (0FFFFh). Any modes found in this list are guaranteed to be available for the current configuration.

The TotalMemory member indicates the maximum amount of memory physically installed and available to the frame buffer in 1Kb units. Note that not all graphics modes will be able to address all of this memory.

The Attributes member contains a number of flags that describes certain important characteristics of the graphics controller. The members are exactly the same as those provided in the *GA\_modeInfo* block for each video mode, but the meaning is slightly different. For each flag defined in the *GA\_AttributeFlagsType* enumeration, it represents whether the controller can support these modes in any available graphics modes. Please see the *GetVideoModeInfo* function for a detailed description of each flags meaning.

The TextSize member contains the size of the text mode framebuffer in bytes. It will generally be 64Kb in length. The TextBasePtr member is a 32-bit physical memory address where the text mode framebuffer memory window is located in the CPU address space. This will generally be 0xB0000 to cover the VGA text framebuffer window (both color and monochrome modes).

The BankSize member contains the size of the banked memory buffer in bytes. It can be either 4Kb or 64Kb in length. The BankedBasePtr member is a 32-bit physical memory address where the banked framebuffer memory window is located in the CPU address space. If the banked framebuffer mode is not available, then this member will be zero.

The LinearSize member is the 32-bit length of the linear frame buffer memory in bytes. It can be any length up to the size of the available video memory. The LinearBasePtr member is the 32-bit physical address of the start of frame buffer memory when the controller is in linear frame buffer memory mode. If the linear framebuffer is not available, then this member will be zero.

The ZBufferSize member is the 32-bit length of the local z-buffer (or depth buffer) memory in bytes. It can be any length up to the size of the available local z-buffer memory. The ZBufferBasePtr member is the 32-bit physical address of the start of local z-buffer memory. Note that if the controller does not have local z-buffer memory, but shares the z-buffer in the local framebuffer memory, these two fields will be set to 0.

The TexMemSize member is the 32-bit length of the local texture memory in bytes. It can be any length up to the size of the available local texture memory. The TexMemBasePtr member is the 32-bit physical address of the start of local texture memory. Note that if the controller does not have local texture memory, but loads textures in the local framebuffer memory, this field will be set to 0.

The LockedMemSize contains the amount of locked, contiguous memory in bytes that the graphics driver requires for programming the hardware. If the graphics accelerator requires DMA transfers for 2D and 3D rendering operations, this member can be set to the length of the block of memory that is required by the driver. The driver loader code will attempt to allocate a block of locked, physically contiguous memory from the operating system and place a pointer to this allocated memory in the LockedMem member for the driver, and the physical address of the start of this memory block in LockedMemPhys. Note that the memory must be locked so it cannot be paged out to disk, and it must be physically contiguous so that DMA operations will work correctly across 4Kb CPU page boundaries. If the driver does not require DMA memory, this value should be set to 0.

The MMIOBase member contains the 32-bit physical base addresses pointing to the start of up to 4 separate memory mapped register areas required by the controller. The MMIOLen member contains the lengths of each of these memory mapped IO areas in bytes. When the application maps the memory mapped IO regions for the driver, the linear address of the mapped memory areas will then be stored in the corresponding entries in the IOMemMaps array, and will be used by the driver for accessing the memory mapped registers on the controller. If any of these regions are not required, the MMIOBase entries will be NULL and do not need to be mapped by the application.

---

**Note:** *The memory regions pointed to by the MMIOBase addresses have special meanings for the first two and second two addresses that are mapped. If the OS loader is running the driver in user space with a safety level of 2, then the only the first two base addresses will be mapped into user space, and the second two will be mapped only into kernel space (kernel space can also access the user space mappings). Please see QueryFunctions for a more detailed overview of the safety levels and how this relates to these regions.*

---

The IOMemMaps member contains the mapped linear address of the memory mapped register regions defined by the MMIOBase and MMIOLen members.

The TextMem member contains the mapped linear address of the text mode framebuffer, and will be filled in by the application when it has loaded the device driver. This provides the device driver with direct access to the video memory on the controller when in text modes.

The BankedMem member contains the mapped linear address of the banked memory framebuffer, and will be filled in by the application when it has loaded the device driver. This provides the device driver with direct access to the video memory on the controller when in the banked framebuffer modes.

The LinearMem member contains the mapped linear address of the linear memory framebuffer, and will be filled in by the application when it has loaded the device driver. This provides the device driver with direct access to the video memory on the controller when in the linear framebuffer modes.

---

**Note:** *On some controllers the linear framebuffer address may be different for different color depths, so the value in this variable may change after initializing a mode. Applications should always reload the address of the linear framebuffer from this variable after initializing a mode set to ensure that the correct value is always used.*

---

The ZBufferMem member contains the mapped linear address of the local z-buffer memory, and will be filled in by the application when it has loaded the device driver. This provides the device driver with direct access to the local z-buffer memory on the controller. If the controller does not have local z-buffer memory, this member will be set to NULL.

The TexBufferMem member contains the mapped linear address of the local texture memory, and will be filled in by the application when it has loaded the device driver. This provides the device driver with direct access to the local texture memory on the controller. If the controller does not have local texture memory, this member will be set to NULL.

The LockedMem member contains a pointer to the locked DMA memory buffer allocated for the loaded driver. The graphics driver can use this pointer to write data directly to the DMA buffer before transferring it to the hardware. If the driver does not require DMA memory, this value will be set to NULL by the loader.

The LockedMemPhys member contains the 32-bit physical memory address of the locked DMA buffer memory allocated for the driver. The graphics driver can use this physical address to set up DMA transfer operations for memory contained within the DMA transfer buffer. If the driver does not require DMA memory, this value will be set to 0 by the loader.

The TextFont8x8, TextFont8x14 and TextFont8x16 members contain pointers to the 8x8, 8x14 and 8x16 text font bitmaps allocated by the OS loader. This data is used by the driver for VGA and extended text modes that require the bitmap font tables.

## Members



<i>Signature</i>	'GRAPHICS\0' 20 byte signature
<i>Version</i>	Driver Interface Version
<i>DriverRev</i>	Driver revision number
<i>OemVendorName</i>	Vendor Name string
<i>OemCopyright</i>	Vendor Copyright string
<i>AvailableModes</i>	Offset to supported mode table
<i>DeviceIndex</i>	Device index for the driver when loaded from disk
<i>TotalMemory</i>	Amount of memory in Kb detected
<i>Attributes</i>	Driver attributes
<i>WorkArounds</i>	Hardware WorkArounds flags
<i>TextSize</i>	Length of the text framebuffer in bytes
<i>TextBasePtr</i>	Base address of the text framebuffer
<i>BankSize</i>	Bank size in bytes (4Kb or 64Kb)
<i>BankedBasePtr</i>	Physical addr of banked buffer
<i>LinearSize</i>	Linear buffer size in bytes
<i>LinearBasePtr</i>	Physical addr of linear buffer
<i>ZBufferSize</i>	Z-buffer size in bytes
<i>ZBufferBasePtr</i>	Physical addr of Z-buffer
<i>TexBufferSize</i>	Texture buffer size in bytes
<i>TexBufferBasePtr</i>	Physical addr of texture buffer
<i>LockedMemSize</i>	Amount of locked memory for driver in bytes
<i>IOBase</i>	Base address for I/O mapped registers (relocateable)
<i>MMIOBase</i>	Base address of memory mapped I/O regions
<i>MMIOLen</i>	Length of memory mapped I/O regions in bytes
<i>DriverStart</i>	Pointer to the start of the driver in memory
<i>DriverSize</i>	Size of the entire driver in memory in bytes
<i>BusType</i>	Indicates the type of bus for the device ( <i>GA_busType</i> )
<i>Attributes</i>	Driver extended attributes flags
<i>IOMemMaps</i>	Pointers to mapped I/O memory
<i>BankedMem</i>	Ptr to mapped banked video mem
<i>LinearMem</i>	Ptr to mapped linear video mem
<i>ZBufferMem</i>	Ptr to mapped zbuffer mem
<i>TexBufferMem</i>	Ptr to mapped texture buffer mem
<i>LockedMem</i>	Ptr to allocated locked memory
<i>LockedMemPhys</i>	Physical addr of locked memory
<i>TextFont8x8</i>	Ptr to 8x8 text font data
<i>TextFont8x14</i>	Ptr to 8x14 text font data
<i>TextFont8x16</i>	Ptr to 8x16 text font data
<i>VGAPal4</i>	Ptr to the default VGA 4bpp palette
<i>VGAPal8</i>	Ptr to the default VGA 8bpp palette
<i>loader</i>	Internal device driver loader functions

## GA\_driverFuncs

**Prototype In**

snap/graphics.h

**Description**

Function group containing all main device driver functions not related to mode initialisation and setup, and not related to drawing and state management. This includes support for things such as changing framebuffer banks, changing the displayed video memory start address and program the hardware palette.

---

**Note:** *Be sure to fill in the dwSize member of this structure when you call GA\_queryFunctions to the correct size of the structure at compile time!*

---

## EnableStereoMode

Enables or disables hardware stereo page flipping.

### Declaration

```
void NAPI GA_driverFuncs::EnableStereoMode(  
    N_int32 enable)
```

### Prototype In

snap/graphics.h

### Parameters

*enable* not 0 to enable stereo mode, 0 to disable

### Description

This function enables or disables hardware stereo LC shutter glasses operation. For stereo LC shutter glasses support, the hardware must provide support for the dual display start addresses so that the left and right images can be located in different locations in display memory. The driver implementation should enable the stereo display when the application calls this function with enable set to 1 and will remain in free running mode until the application calls this function again with enable set to 0. Check that the gaHaveStereo flag is set in the *GA\_modeInfo* structure before trying to use this function.

### See Also

*SetStereoDisplayStart, GetDisplayStartStatus, SetActiveBuffer, FlipToStereoBuffer, GetFlipStatus*

## GetCurrentScanLine

Returns the current vertical scanline that the hardware is displaying.

### Declaration

```
N_int32 NAPI GA_driverFuncs::GetCurrentScanLine(void)
```

### Prototype In

snap/graphics.h

### Return Value

Current scanline being displayed by the hardware.

### Description

This function reads the hardware to determine what scanline is currently being displayed at the time of the call. This can be used to determine where on the screen the CRT raster beam is located, for special effects such as beam following animation.

---

**Note:** *Note that not all hardware supports this functionality. If the hardware does not support this, this function will be a NULL pointer.*

---

## GetDisplayStartStatus

Returns the status of the last scheduled display start change.

### Declaration

```
N_int32 NAPI GA_driverFuncs::GetDisplayStartStatus(void)
```

### Prototype In

snap/graphics.h

### Return Value

0 if flip has not occurred, not 0 if it has

### Description

This function returns the status of the last scheduled display start change set if SetVisibleBuffer was called with the waitVRT flag set to 0.

### See Also

*SetDisplayStart, SetActiveBuffer, FlipToBuffer, GetFlipStatus*

## GetGammaCorrectData

Returns the current hardware gamma correction table.

### Declaration

```
void NAPI GA_driverFuncs::GetGammaCorrectData (
    GA_palette *pal,
    N_int32 num,
    N_int32 index)
```

### Prototype In

snap/graphics.h

### Parameters

<i>pal</i>	Place to return the gamma data
<i>num</i>	Number of gamma entries to read
<i>index</i>	Index of first entry to read

### Description

This function reads the hardware gamma correction tables for 15 bit and above graphics modes. The gamma correction tables are used in these graphics modes to adjust the response curves of each of the three color guns for color matching purposes. The gamma correction tables are assumed to be 256 entries deep with three independent channels for each of red, green and blue. Each value in the gamma tables are 8-bits wide, with a range of 0 to 255. Gamma correction data is passed to the function in an array of *GA\_palette* structures, similar to the *SetPaletteData* function.

---

**Note:** *If this hardware does not support gamma correction, this function will be a NULL pointer.*

---

### See Also

*SetPaletteData, GetPaletteData, SetGammaCorrectData, GetGammaCorrectDataExt, SetPaletteDataExt, GetPaletteDataExt, SetGammaCorrectDataExt*

## GetGammaCorrectDataExt

Returns the current hardware gamma correction table using 16-bit per color channel

### Declaration

```
void NAPI GA_driverFuncs::GetGammaCorrectDataExt (
    GA_paletteExt *pal,
    N_int32 num,
    N_int32 index)
```

### Prototype In

snap/graphics.h

### Parameters

<i>pal</i>	Place to return the gamma data
<i>num</i>	Number of gamma entries to read
<i>index</i>	Index of first entry to read

### Description

This function reads the hardware gamma correction tables for 15 bit and above graphics modes. The gamma correction tables are used in these graphics modes to adjust the response curves of each of the three color guns for color matching purposes. The gamma correction tables are assumed to be 256 entries deep with three independent channels for each of red, green and blue. Each value in the gamma tables are 16-bits wide, with a range of 0 to 65535. Note that this is different to the regular *GetGammaCorrectData* function, which takes 8-bit wide values. Internally the driver will convert the 16-bit palette values to 8-bits if this is what the underlying hardware supports.

---

**Note:** *If this hardware does not support gamma correction, this function will be a NULL pointer.*

---

### See Also

*SetPaletteDataExt, GetPaletteDataExt, SetGammaCorrectDataExt, GetGammaCorrectData, SetPaletteData, GetPaletteData, SetGammaCorrectData*

## GetPaletteData

Returns the current hardware color palette.

### Declaration

```
void NAPI GA_driverFuncs::GetPaletteData(
    GA_palette *pal,
    N_int32 num,
    N_int32 index)
```

### Prototype In

snap/graphics.h

### Parameters

<i>pal</i>	Place to return the palette data
<i>num</i>	Number of palette entries to read
<i>index</i>	Index of first entry to read

### Description

This function reads the hardware color palette information from the hardware, and is only valid in 8-bpp and lower color index modes. Color palette information is returned from this function in an array of *GA\_palette* structures. Each value in the *GA\_palette* structure is 8-bits wide, with a range of 0 to 255. Note that this is different to the standard VGA palette programming routines, which normally take 6-bit wide values. Internally the driver will convert the 8-bit palette values to 6-bits if this is what the underlying hardware supports.

### See Also

*SetPaletteData*, *SetGammaCorrectData*, *GetGammaCorrectData*, *GetPaletteDataExt*, *SetPaletteDataExt*, *SetGammaCorrectDataExt*, *GetGammaCorrectDataExt*



## GetPaletteDataExt

Returns the current hardware color palette using 16-bit per color channel

### Declaration

```
void NAPI GA_driverFuncs::GetPaletteDataExt (
    GA_paletteExt *pal,
    N_int32 num,
    N_int32 index)
```

### Prototype In

snap/graphics.h

### Parameters

<i>pal</i>	Place to return the extended palette data
<i>num</i>	Number of palette entries to read
<i>index</i>	Index of first entry to read

### Description

This function reads the hardware color palette information from the hardware, and is only valid in 8-bpp and lower color index modes. Color palette information is returned from this function in an array of *GA\_paletteExt* structures. Each value in the *GA\_paletteExt* structure is 16-bits wide, with a range of 0 to 65535. Note that this is different to the regular *SetPaletteData* function, which takes 8-bit wide values. Internally the driver will convert the 16-bit palette values to 8-bits if this is what the underlying hardware supports.

### See Also

*SetPaletteDataExt*, *SetGammaCorrectDataExt*, *GetGammaCorrectDataExt*, *GetPaletteData*, *SetPaletteData*, *SetGammaCorrectData*, *GetGammaCorrectData*

## GetVSyncWidth

Returns the current vertical sync width.

### Declaration

```
N_int32 NAPI GA_driverFuncs::GetVSyncWidth(void)
```

### Prototype In

snap/graphics.h

### Return Value

Current vertical sync width of CRTC display mode

### Description

This function returns the current vertical sync width for the CRTC controller for the current display mode. You can only call this function once you have called SetVideoMode to initialise a valid display mode.

### See Also

*SetVSyncWidth*

## IsVSync

Determines if the card is currently within the vertical retrace interval.

### Declaration

```
N_int32 NAPI GA_driverFuncs::IsVSync(void)
```

### Prototype In

snap/graphics.h

### Return Value

not 0 if currently in the vertical sync interval, 0 if not.

### Description

This function determines if the CRTC controller is currently within the vertical retrace interval or not.

### See Also

*WaitVSync*

## SetBank

Set the current bank for banked framebuffer rendering.

### Declaration

```
void NAPI_GA_driverFuncs::SetBank(  
    N_int32 bank)
```

### Prototype In

snap/graphics.h

### Parameters

*bank*                New bank to make the active read/write bank

### Description

This function changes the currently active read/write bank for banked framebuffer modes. This allows an application to directly access the video framebuffer through a small memory aperture window. Only a single read/write bank is supported, and it may be either 4Kb or 64Kb in length.

## SetDisplayStart

Sets the currently visible display start address.

### Declaration

```
void NAPI_GA_driverFuncs::SetDisplayStart(
    N_int32 offset,
    N_int32 waitVRT)
```

### Prototype In

snap/graphics.h

### Parameters

<i>offset</i>	Offset to start of display memory to make visible
<i>waitVRT</i>	Wait for retrace flag (true or false)

### Description

This function sets the currently visible hardware display start address as a byte offset from the start of physical display memory. Hence to display data beginning at the start of video memory you would set the offset parameter to 0. To display the second page for double buffered animation you would set the offset parameter to 'YResolution \* BytesPerScanLine' for the current video mode. The waitVRT flag determines if the function will wait for the vertical retrace when programming the hardware display start address and the following are valid values:

<i>value</i>	Description
0	Schedule change for next retrace and return immediately not 0 - Wait for vertical retrace during programming

Generally you need to wait for a vertical retrace to enable flicker free animation when doing double buffered animation. However if you have set up three visible display buffers for hardware triple buffering (check that the gaHaveTripleBuffer flag is set in the *GA\_modeInfo* structure) and you pass a value of 0 for waitVRT. In this case the function will schedule the display start address and return immediately, and you can then call the *GetDisplayStartStatus* function at a later date to determine if the previous display start address has taken hold yet or not. In order for this to work properly, before you need to access the next buffer for rendering, you should also loop until *GetDisplayStartStatus* indicates that the last flip has taken place (by default this will always be the case when a mode is first initialized).

---

**Note:** *The value passed into this function is always in units of bytes for 8bpp and above display modes (and text modes). For 4bpp display modes the value is defined in units of pixels so that pixel perfect scrolling can be achieved.*

**Note:** *If you have enabled the buffer manager, please use the buffer manager function FlipToBuffer to change the display start address instead.*

---

### See Also

*SetDisplayStartXY, SetStereoDisplayStart, GetDisplayStartStatus, SetActiveBuffer,  
FlipToBuffer*

## SetDisplayStartXY

Sets the currently visible display start address as an (x,y) coordinate.

### Declaration

```
void NAPI_GA_driverFuncs::SetDisplayStartXY(
    N_int32 x,
    N_int32 y,
    N_int32 waitVRT)
```

### Prototype In

snap/graphics.h

### Parameters

<i>x</i>	X coordinate of first pixel to display
<i>y</i>	Y coordinate of first pixel to display
<i>waitVRT</i>	Wait for retrace flag

### Description

This function sets the currently visible hardware display start address as an (x,y) coordinate offset from the start of physical display memory. This is similar to *SetDisplayStart*, however the driver does the conversion from an (x,y) coordinate to the display memory offset for you. The waitVRT flag determines if the function will wait for the vertical retrace when programming the hardware display start address and the following are valid values:

<i>value</i>	Description
0	Schedule change for next retrace and return immediately not 0 - Wait for vertical retrace during programming

### See Also

*SetDisplayStart*, *SetStereoDisplayStart*, *GetDisplayStartStatus*, *SetActiveBuffer*, *FlipToBuffer*

## SetGammaCorrectData

Programs the hardware gamma correction table.

### Declaration

```
void NAPI_GA_driverFuncs::SetGammaCorrectData (
    GA_palette *pal,
    N_int32 num,
    N_int32 index,
    N_int32 waitVRT)
```

### Prototype In

snap/graphics.h

### Parameters

<i>pal</i>	Pointer to the gamma data to program
<i>num</i>	Number of gamma entries to program
<i>index</i>	Index of first entry to program
<i>waitVRT</i>	Wait for vertical retrace flag

### Description

This function programs the gamma correction tables for 15 bit and above graphics modes. The gamma correction tables are used in these graphics modes to adjust the response curves of each of the three color guns for color matching purposes. The gamma correction tables are assumed to be 256 entries deep with three independent channels for each of red, green and blue. Each value in the gamma tables are 8-bits wide, with a range of 0 to 255. Gamma correction data is passed to the function in an array of *GA\_palette* structures, similar to the *SetPaletteData* function.

The wait for vertical retrace flag is used to synchronize the palette update with the start of the vertical retrace. The following are valid values:

<i>value</i>	Description
0	Change palette immediately not 0 - Program palette during vertical retrace period

However if you are changing palette values at the same time as swapping display pages, you may want to disable vertical retrace synching and program the palette entries directly after swapping display pages. Generally you need to synchronize with the vertical retrace while programming the palette to avoid the onset of snow (or interference on the screen).

---

**Note:** *If this hardware does not support gamma correction, this function will be a NULL pointer.*

---

### See Also

*GetPaletteData, SetPaletteData, GetGammaCorrectData, SetGammaCorrectDataExt, GetPaletteDataExt, SetPaletteDataExt, GetGammaCorrectDataExt*



## SetGammaCorrectDataExt

Programs the hardware gamma correction table using 16-bit per color channel

### Declaration

```
void NAPI GA_driverFuncs::SetGammaCorrectDataExt (
    GA_paletteExt *pal,
    N_int32 num,
    N_int32 index,
    N_int32 waitVRT)
```

### Prototype In

snap/graphics.h

### Parameters

<i>pal</i>	Pointer to the gamma data to program
<i>num</i>	Number of gamma entries to program
<i>index</i>	Index of first entry to program
<i>waitVRT</i>	Wait for vertical retrace flag

### Description

This function programs the gamma correction tables for 15 bit and above graphics modes. The gamma correction tables are used in these graphics modes to adjust the response curves of each of the three color guns for color matching purposes. The gamma correction tables are assumed to be 256 entries deep with three independent channels for each of red, green and blue. Each value in the gamma tables are 16-bits wide, with a range of 0 to 65535. Note that this is different to the regular *SetGammaCorrectData* function, which takes 8-bit wide values. Internally the driver will convert the 16-bit palette values to 8-bits if this is what the underlying hardware supports.

This function is basically the same as *SetGammaCorrectData* except it deals with a gamma ramp using 16-bit values per channel.

---

**Note:** *If this hardware does not support gamma correction, this function will be a NULL pointer.*

---

### See Also

*GetPaletteDataExt, SetPaletteDataExt, GetGammaCorrectDataExt, SetGammaCorrectData, GetPaletteData, SetPaletteData, GetGammaCorrectData*

## SetPaletteData

Programs the hardware color palette.

### Declaration

```
void NAPI GA_driverFuncs::SetPaletteData(
    GA_palette *pal,
    N_int32 num,
    N_int32 index,
    N_int32 waitVRT)
```

### Prototype In

snap/graphics.h

### Parameters

<i>pal</i>	Pointer to the palette data to program
<i>num</i>	Number of palette entries to program
<i>index</i>	Index of first entry to program
<i>waitVRT</i>	Wait for vertical retrace flag

### Description

This function programs the color palette information for the current graphics mode, and is only valid in 8-bpp and lower color index modes. Color palette information is passed to the function in an array of *GA\_palette* structures. Each value in the *GA\_palette* structure is 8-bits wide, with a range of 0 to 255. Note that this is different to the standard VGA palette programming routines, which normally take 6-bit wide values. Internally the driver will convert the 8-bit palette values to 6-bits if this is what the underlying hardware supports.

The wait for vertical retrace flag is used to synchronize the palette update with the start of the vertical retrace. The following are valid values:

<i>value</i>	Description
0	Change palette immediately not 0 - Program palette during vertical retrace period

However if you are changing palette values at the same time as swapping display pages, you may want to disable vertical retrace synching and program the palette entries directly after swapping display pages. Generally you need to synchronize with the vertical retrace while programming the palette to avoid the onset of snow (or interference on the screen).

### See Also

*GetPaletteData*, *SetGammaCorrectData*, *GetGammaCorrectData*, *SetPaletteDataExt*, *GetPaletteDataExt*, *SetGammaCorrectDataExt*, *GetGammaCorrectDataExt*

## SetPaletteDataExt

Programs the hardware color palette using 16-bit per color channel

### Declaration

```
void NAPI GA_driverFuncs::SetPaletteDataExt (
    GA_paletteExt *pal,
    N_int32 num,
    N_int32 index,
    N_int32 waitVRT)
```

### Prototype In

snap/graphics.h

### Parameters

<i>pal</i>	Pointer to the extended palette data to program
<i>num</i>	Number of palette entries to program
<i>index</i>	Index of first entry to program
<i>waitVRT</i>	Wait for vertical retrace flag

### Description

This function programs the color palette information for the current graphics mode, and is only valid in 8-bpp and lower color index modes. Color palette information is passed to the function in an array of *GA\_paletteExt* structures. Each value in the *GA\_paletteExt* structure is 16-bits wide, with a range of 0 to 65535. Note that this is different to the regular *SetPaletteData* function, which takes 8-bit wide values. Internally the driver will convert the 16-bit palette values to 8-bits if this is what the underlying hardware supports.

This function is basically the same as *SetPaletteData* except it deals with a palette with 16-bit color values per channel.

### See Also

*GetPaletteDataExt*, *SetGammaCorrectDataExt*, *GetGammaCorrectDataExt*, *SetPaletteData*, *GetPaletteData*, *SetGammaCorrectData*, *GetGammaCorrectData*

## SetStereoDisplayStart

Sets the currently visible stereo display start address.

### Declaration

```
void NAPI_GA_driverFuncs::SetStereoDisplayStart (
    N_int32 leftOffset,
    N_int32 rightOffset,
    N_int32 waitVRT)
```

### Prototype In

snap/graphics.h

### Parameters

<i>leftOffset</i>	Offset to start of left display image (in bytes)
<i>rightOffset</i>	Offset to start of right display image (in bytes)
<i>waitVRT</i>	Wait for retrace flag

### Description

This function is identical to the *SetDisplayStart* function except that it takes both left and right display start address offsets. If the display controller supports a hardware stereo display mode, it will alternate between displaying the left image and right image every vertical retrace. This function is used to program the left and right display start addresses to different values for when hardware stereo mode is enabled. Check that the *gaHaveStereo* flag is set in the *GA\_modelInfo* structure before trying to use this function. The *waitVRT* flag determines if the function will wait for the vertical retrace when programming the hardware display start address and the following are valid values:

<i>value</i>	Description
0	Schedule change for next retrace and return immediately not 0 - Wait for vertical retrace during programming

---

**Note:** *The value passed into this function is always in units of bytes for 8bpp and above display modes (and text modes). For 4bpp display modes the value is defined in units of pixels so that pixel perfect scrolling can be achieved.*

---

### See Also

*SetDisplayStart*, *GetDisplayStartStatus*, *EnableStereoMode*, *SetActiveBuffer*, *FlipToStereoBuffer*

## SetVSyncWidth

Set the current vertical sync width.

### Declaration

```
void NAPI_GA_driverFuncs::SetVSyncWidth(  
    N_int32 width)
```

### Prototype In

snap/graphics.h

### Parameters

*width*                      New vertical sync width to program

### Description

This function changes the current vertical sync width for the CRTC controller to the passed in sync width. You should use this function with care, and only program values that are within +-1 from the original sync width returned by *GetVSyncWidth*. This function is intended primarily to support stereo LC shutter glasses that use the sync width to determine the left and right frames of the stereo display image.

### See Also

*GetVSyncWidth*

## WaitVSync

Waits until the graphics card enters the vertical retrace interval.

### Declaration

```
void NAPI GA_driverFuncs::WaitVSync(void)
```

### Prototype In

snap/graphics.h

### Description

This function waits until the CRTC controller is enters the start of the vertical retrace interval and then returns.

### See Also

*IsVSync*

## GA\_funcGroupsType

### Declaration

```
typedef enum {
    GA_GET_RESERVED,
    GA_GET_INITFUNCS,
    GA_GET_DRIVERFUNCS,
    GA_GET_CURSORFUNCS,
    GA_GET_VIDEOfUNCS,
    GA_GET_DPMSFUNCS,
    GA_GET_SCIFUNCS,
    GA_GET_2DSTATEFUNCS,
    GA_GET_2DRENDERFUNCS,
    GA_GET_3DSETUPFUNCS,
    GA_GET_3DSTATEFUNCS,
    GA_GET_3DRENDERFUNCS,
    GA_GET_D3DRENDERFUNCS,
    GA_GET_VBEFUNCS,
    GA_GET_REGIONFUNCS,
    GA_GET_BUFFERFUNCS,
    GA_GET_CLIPPERFUNCS,
    GA_GET_FIRST_OEM                = 0x00010000
} GA_funcGroupsType
```

### Prototype In

snap/graphics.h

### Description

This enumeration defines the identifiers used to obtain the device context function group pointer structures. As new features and capabilities are added to the future versions of the specification, new identifiers will be added to extract new function pointers from the drivers.

The GA\_GET\_FIRST\_OEM defines the first identifier for OEM extensions. OEM's are free to add their own private functional extensions to the drivers as desired. Note that OEM's must verify the presence of their OEM drivers via the `OemVendorName` string before attempting to use OEM extension functions.

### Members

<b>GA_GET_RESERVED</b>	Reserved value
<b>GA_GET_INITFUNCS</b>	Get <i>GA_initFuncs</i> structure
<b>GA_GET_DRIVERFUNCS</b>	Get <i>GA_driverFuncs</i> structure
<b>GA_GET_CURSORFUNCS</b>	Get <i>GA_cursorFuncs</i> structure
<b>GA_GET_VIDEOfUNCS</b>	Get <i>GA_videoFuncs</i> structure
<b>GA_GET_DPMSFUNCS</b>	Get <i>GA_DPMSFuncs</i> structure
<b>GA_GET_SCIFUNCS</b>	Get <i>GA_SCIFuncs</i> structure
<b>GA_GET_2DSTATEFUNCS</b>	Get <i>GA_2DStateFuncs</i> structure
<b>GA_GET_2DRENDERFUNCS</b>	Get <i>GA_2DRenderFuncs</i> structure
<b>GA_GET_3DSETUPFUNCS</b>	Get <i>GA_3DSetupFuncs</i> structure
<b>GA_GET_3DSTATEFUNCS</b>	Get <i>GA_3DStateFuncs</i> structure
<b>GA_GET_3DRENDERFUNCS</b>	Get <i>GA_3DRenderFuncs</i> structure
<b>GA_GET_D3DRENDERFUNCS</b>	Get <i>GA_D3DRenderFuncs</i> structure

<i>GA_GET_VBEFUNCS</i>	Get <i>GA_VBEFuncs</i> structure
<i>GA_GET_REGIONFUNCS</i>	Get <i>GA_regionFuncs</i> structure
<i>GA_GET_BUFFERFUNCS</i>	Get <i>GA_bufferFuncs</i> structure
<i>GA_GET_CLIPPERFUNCS</i>	Get <i>GA_clipperFuncs</i> structure
<i>GA_GET_FIRST_OEM</i>	ID of first OEM extension function



## GA\_globalOptions

### Declaration

```
typedef struct {
    N_uint32      dwSize;
    N_uint8      bVirtualDisplay;
    N_uint8      bPortrait;
    N_uint8      bFlipped;
    N_uint8      bInvertColors;
    N_uint8      bVBEOnly;
    N_uint8      bVGAOnly;
    N_uint8      bReserved1;
    N_uint16     wCertifiedVersion;
    N_uint8      bNoWriteCombine;
    N_uint8      bAllowNonCertified;
    N_uint8      bLCDUseBIOS;
    N_uint8      bUseMemoryDriver;
    N_uint16     wSysMemSize;
    N_uint32     dwReserved2;
    N_uint8      bVBEUseLinear;
    N_uint8      bVBEUsePal;
    N_uint8      bVBEUsePM32;
    N_uint8      bReserved2;
    N_uint8      bVBEUseVBE20;
    N_uint8      bVBEUseVBE30;
    N_uint8      bVBEUsePM;
    N_uint8      bVBEUseSCI;
    N_uint8      bVBEUseDDC;
    N_uint8      bGDIUseAccel;
    N_uint8      bGDIUseBrushCache;
    N_uint8      bGDIUseBitmapCache;
    N_uint8      bDXUseAccel2D;
    N_uint8      bDXUseAccel3D;
    N_uint8      bDXUseAccelVideo;
    N_uint8      bDXWaitRetrace;
    N_uint32     dwCPLFlags;
    N_uint32     dwSharedAGPMemSize;
    N_uint8      bUseVBECORE;
    N_uint8      bUseVGACORE;
    N_uint32     dwCheckForUpdates;
    N_uint8      bNoDDCDetect;
    N_uint8      bDisableLogFile;
    N_uint8      bCheckWebSelection;
    N_uint16     wMonitorHSize;
    N_uint16     wMonitorVSize;
    N_uint16     wOptimizedModeXRes;
    N_uint16     wOptimizedModeYRes;
    N_uint16     wOptimizedModeBits;
    GA_recMode   recommendedMode;
    GA_recMode   recommendedMode8;
    GA_recMode   recommendedMode16;
    GA_recMode   recommendedMode24;
    GA_recMode   recommendedMode32;
    N_uint8      bAGPFastWrite;
    N_uint8      res1[67];
    GA_layout    virtualSize;
    GA_layout    resolutions[GA_MAX_VIRTUAL_DISPLAYS];
    GA_layout    bounds[GA_MAX_VIRTUAL_DISPLAYS];
} GA_globalOptions
```

### Prototype In

snap/graphics.h

### Description

Structure returned by *GA\_getGlobalOptions*, which contains configuration information about the options effective for all installed display devices. This structure also contains the layout information used for multi-controller options in SNAP Graphics (such as what screen is located where).

---

**Note:** *The dwSize member is intended for future compatibility, and should be set to the size of the structure as defined in this header file. Future drivers will be compatible with older software by examining this value.*

---

### Members

<i>dwSize</i>	Set to size of structure in bytes
<i>bVirtualDisplay</i>	Enable virtual display mode
<i>bPortrait</i>	Enable portrait display mode
<i>bFlipped</i>	Enable flipped display mode
<i>bInvertColors</i>	Enable invert color mode
<i>bVBEOnly</i>	Enable VBE/Core fallback driver
<i>bVGAOnly</i>	Enable VGA fallback driver
<i>bReserved1</i>	Reserved option; must <b>always</b> be zero!
<i>bAllowNonCertified</i>	Allow uncertified drivers to load
<i>wCertifiedVersion</i>	Version of certify program to allow
<i>bNoWriteCombine</i>	Disable write combining
<i>bLCDUseBIOS</i>	Enable use of BIOS when on the LCD panel
<i>bUseMemoryDriver</i>	Enable system memory driver
<i>wSysMemSize</i>	Amount of memory to allocate for sysmem driver (Kb)
<i>dwCPLFlags</i>	Place to store control panel UI settings
<i>dwSharedAGPMemSize</i>	Amount of shared AGP memory to use for framebuffer
<i>bUseVBECORE</i>	Use the VBE/Core emulation driver
<i>bUseVGACORE</i>	Use the VGA/Core emulation driver
<i>dwCheckForUpdates</i>	Time stamp to check for updates next
<i>bNoDDCdetect</i>	Disable automatic DDC monitor detection
<i>bDisableLogFile</i>	Disable logging of information to log file
<i>bCheckWebSelection</i>	SDD GUI specific value for web check updates
<i>virtualSize</i>	Virtual size for multi-controller displays
<i>resolutions</i>	Physical resolutions for multi-controller displays
<i>bounds</i>	Virtual layout for multi-controller displays

## GA\_initFuncs

**Prototype In**

snap/graphics.h

**Description**

Function group containing all device driver init functions available for the device. These functions include all mode information, setup and initialisation functions.

---

**Note:** *Be sure to fill in the dwSize member of this structure when you call GA\_queryFunctions to the correct size of the structure at compile time!*

---

## AlignLinearBuffer

Aligns the linear start address to a hardware required boundary

### Declaration

```
ibool NAPI GA_initFuncs::AlignLinearBuffer(
    N_int32 height,
    N_int32 *stride,
    N_int32 *offset,
    N_int32 *size,
    N_int32 growUp)
```

### Prototype In

snap/graphics.h

### Parameters

<i>height</i>	Height of the buffer to align
<i>stride</i>	Stride of the buffer to align (modified)
<i>offset</i>	Starting offset of the buffer to align (modified)
<i>size</i>	Place to return the size of the resulting buffer
<i>growUp</i>	True if the buffer is allocated on a heap that grows up in memory

### Return Value

True if the buffer was successfully aligned in video memory, false if not.

### Description

This optional function is used to align a buffer in offscreen video memory as necessary for the hardware. If this function is not implemented, it is assumed the hardware can work with simple fixed alignment requirements, and the buffers should be aligned as per the alignment boundaries described by the `BitmapStartAlign` and `BitmapStridePad` members of the `GA_modeInfo` structure. However some hardware has special alignment requirements that cannot be easily described with a simple fixed alignment factor, so in those cases this function will need to be called to align the offscreen memory buffer appropriately.

If the `growUp` flag is true, the buffer is aligned such that the memory grows up in memory above the initial value passed in the `offset` parameter. If the `growUp` flag is false, the buffer is aligned such that the memory grows down from the initial value passed in the `offset` parameter. The value returned in the `offset` parameter is always the value at the start of the buffer in memory (ie: lowest memory address).

## GetActiveHead

Return the currently active output head for the device

### Declaration

```
N_int32 NAPI GA_initFuncs::GetActiveHead(void)
```

### Prototype In

snap/graphics.h

### Return Value

Index of currently active head (*GA\_multiHeadType*)

### Description

This function is determine the currently active output head for the device.

### See Also

*GetNumberOfHeads*, *SetActiveHead*

## GetCRTCTimings

Returns the current CRTC timings for the active display mode.

### Declaration

```
void NAPI GA_initFuncs::GetCRTCTimings(  
    GA_CRTCInfo *crtc)
```

### Prototype In

snap/graphics.h

### Parameters

*crtc*                      Place to store the active CRTC timings

### Description

This function returns a copy of the currently active CRTC timings for the active display mode in the driver. This function is mostly used for interactive centering and refresh control in utility programs.

### See Also

*SetCRTCTimings, GetCurrentRefreshRate, SetGlobalRefresh, GA\_saveCRTCTimings, GA\_restoreCRTCTimings, GA\_getCRTCTimings, GA\_setCRTCTimings, GA\_setDefaultRefresh*

## GetCertifyInfo

Returns the current certification information block for the driver

### Declaration

```
void NAPI GA_initFuncs::GetCertifyInfo(  
    GA_certifyInfo *info)
```

### Prototype In

snap/graphics.h

### Parameters

*info*                      Place to store the returned certification information

### Description

This function returns a structure which contains complete certification information about the loaded device driver. The information contained in this structure is informational only, and intened mainly for technical support purposes.

---

**Note:** *The dwSize member of the profile structure is intended for future compatibility, and must be set to the size of the structure before calling this function. Only the number of bytes set in the dwSize member will be copied into the callers structure.*

---

### See Also

*GetConfigInfo*

## GetClosestPixelClock

Finds the closest pixel clock to the requested pixel clock.

### Declaration

```
N_uint32 NAPI GA_initFuncs::GetClosestPixelClock(
    N_int32 xRes,
    N_int32 yRes,
    N_int32 bitsPerPixel,
    N_uint32 pixelClock)
```

### Prototype In

snap/graphics.h

### Parameters

<i>xRes</i>	Physical X resolution for the display mode
<i>yRes</i>	Physical Y resolution for the display mode
<i>bitsPerPixel</i>	Color depth for the display mode
<i>pixelClock</i>	Requested pixel clock in units of Hz.

### Return Value

Closest pixel clock in units of Hz.

### Description

This function allows an application to determine if a particular pixel clock is available. When this function is called it will run the requested pixel clock through the internal PLL programming routines and return the actual pixel clock that will be programmed into the hardware. The process of running the PLL clock computation routines may cause the returned pixel clock to be rounded slightly up or down from the requested value, however the driver should implement the algorithms to attempt to find clocks that are the same as or higher than the requested value. Note that the calling application must also pass in the physical display resolution and color depth for the mode that will be using this pixel clock to this function. This information is necessary so that the driver can determine any necessary hardware limitations internally for the display mode before looking for the closest physical pixel clock.

If the driver implementation uses a table driven clock programming approach, it should always attempt to find the next highest pixel clock in the table to the requested clock. The exception to this is if there is a lower clock in the table that is within a tolerance of 1% of the requested clock in which case this clock should be returned (and the next highest pixel clock is not within 1% of the requested clock).

This pixel clock can then be used by the application to compute the exact GTF CRTC timing parameters for the mode. Note that for hardware that is not fully programmable, the returned pixel clock that is the closest the one desired may be substantially different (ie: you could get back 39Mhz when you request 35Mhz). It is up to the calling application to determine if the clock is suitable and to attempt to choose a different clock if not suitable. The pixel clocks passed in and returned occupy 32-bits and represents the pixel



clock in units of Hz (ie: a pixel clock of 25.18Mhz is represented with a value of 25180000).

**See Also**

*SetVideoMode, SetCustomVideoMode, GetVideoModeInfo, GetCustomVideoModeInfo*

## GetConfigInfo

Returns information about the installed graphics device.

### Declaration

```
void NAPI GA_initFuncs::GetConfigInfo(  
    GA_configInfo *info)
```

### Prototype In

snap/graphics.h

### Parameters

*info*                    Place to store the returned hardware configuration information

### Description

This function returns a structure defining the hardware configuration information for the installed graphics device that the loaded driver is controlling. This information is informative and only intended for debugging and technical support.

---

**Note:** *The dwSize member of the info structure is intended for future compatibility, and must be set to the size of the structure before calling this function. Only the number of bytes set in the dwSize member will be copied into the callers structure.*

---

### See Also

*GetCertifyInfo*

## GetCurrentRefreshRate

Returns the currently active refresh rate for the display mode

### Declaration

```
N_int32 NAPI GA_initFuncs::GetCurrentRefreshRate(void)
```

### Prototype In

snap/graphics.h

### Return Value

Currently active refresh rate units of 0.01 Hz

### Description

This function returns the currently active refresh rate for the display mode is units of 0.01 Hz (ie: 60.5 Hz is a value 605).

### See Also

*SetVideoMode, GetVideoModeInfoExt, GetCustomVideoModeInfo, GetCustomVideoModeInfoExt, GetCurrentVideoModeInfo,*

## GetCurrentVideoModeInfo

Returns information about the current display mode.

### Declaration

```
void NAPI_GA_initFuncs::GetCurrentVideoModeInfo(
    GA_modeInfo *modeInfo)
```

### Prototype In

snap/graphics.h

### Parameters

*modeInfo*                      Place to store the current mode information (*GA\_modeInfo*)

### Description

This function returns complete information about the currently active SNAP display mode. This function fills the *GA\_modeInfo* structure with detailed information about the current mode, and will include correct information about the virtual resolution, scanline width etc. Hence the values returned here will accurately reflect any changes made with the call to *SetVideoMode* so the mode information may be slightly different to what you would get if you called *GetVideoModeInfo* for the mode number returned by *GetVideoMode*.

---

**Note:** *The calling code must first ensure that the dwSize member of the GA\_modeInfo structure is set to the size of the structure in bytes before calling this function. Only the number of bytes set in the dwSize member will be copied into the callers structure.*

---

### See Also

*GetVideoModeInfo*, *GetVideoModeInfoExt*, *GetCustomVideoModeInfo*,  
*GetCustomVideoModeInfoExt*, *GetCurrentRefreshRate*, *SetVideoMode*

## GetCustomVideoModeInfo

Returns information about a custom display mode.

### Declaration

```
N_int32 NAPI GA_initFuncs::GetCustomVideoModeInfo(
    N_int32 xRes,
    N_int32 yRes,
    N_int32 virtualX,
    N_int32 virtualY,
    N_int32 bitsPerPixel,
    GA_modeInfo *modeInfo)
```

### Prototype In

snap/graphics.h

### Parameters

<i>xRes</i>	Physical X resolution for the display mode
<i>yRes</i>	Physical Y resolution for the display mode
<i>virtualX</i>	Logical X resolution for the display mode
<i>virtualY</i>	Logical Y resolution for the display mode
<i>bitsPerPixel</i>	Color depth for the display mode
<i>modeInfo</i>	Place to store the returned mode information ( <i>GA_modeInfo</i> )

### Return Value

0 on success, -1 on failure

### Description

This function returns extended information about a custom SNAP display mode. A custom display mode does not need to exist in the mode list pointed to by the AvailableModes pointer in the *GA\_devCtx* structure, and this function will fail for modes that the hardware cannot handle. If the requested mode can be handled by the hardware, this function then fills in the *GA\_modeInfo* structure with detailed information about the requested custom display mode.

---

**Note:** Internally SNAP driver have no concept for real display modes, so the regular *GetVideoModeInfo* function also ends up calling this function internally for modes lists in the driver mode profile.

**Note:** The calling code must first ensure that the *dwSize* member of the *GA\_modeInfo* structure is set to the size of the structure in bytes before calling this function. Only the number of bytes set in the *dwSize* member will be copied into the callers structure.

---

### See Also

*GetVideoModeInfo*, *SetVideoMode*, *SetCustomVideoMode*

## GetCustomVideoModeInfoExt

Returns information about a custom display mode for a specific output device

### Declaration

```
N_int32 NAPI GA_initFuncs::GetCustomVideoModeInfoExt (
    N_int32 xRes,
    N_int32 yRes,
    N_int32 virtualX,
    N_int32 virtualY,
    N_int32 bitsPerPixel,
    GA_modeInfo *modeInfo,
    N_int32 outputDevice)
```

### Prototype In

snap/graphics.h

### Parameters

<i>xRes</i>	Physical X resolution for the display mode
<i>yRes</i>	Physical Y resolution for the display mode
<i>virtualX</i>	Logical X resolution for the display mode
<i>virtualY</i>	Logical Y resolution for the display mode
<i>bitsPerPixel</i>	Color depth for the display mode
<i>modeInfo</i>	Place to store the returned mode information ( <i>GA_modeInfo</i> )
<i>outputDevice</i>	Output device flags to use ( <i>GA_OutputFlagsType</i> )

### Return Value

0 on success, -1 on failure

### Description

This function returns extended information about a custom SNAP display mode for a specific output device selection. This is different to the normal *GetCustomVideoModeInfo* function, which returns the display mode information for the currently active output device (ie: LCD, CRT or TV). The only difference is the addition of the *outputDevice* parameter, which contains flags from the *GA\_OutputFlagsType* enumeration.

This function is useful if you are running on a different output device (ie: CRT display) but wish to find out the capabilities of another output device (ie: LCD or TV), without needing to first switch to that device and make it active.

---

**Note:** *The calling code must first ensure that the dwSize member of the GA\_modeInfo structure is set to the size of the structure in bytes before calling this function. Only the number of bytes set in the dwSize member will be copied into the callers structure.*

---

### See Also

*GetVideoModeInfo, GetVideoModeInfoExt, GetCustomVideoModeInfo, GetCurrentVideoModeInfo, GetCurrentRefreshRate, SetVideoMode*

## GetDisplayOutput

Returns the currently active device(s) for display output

### Declaration

```
N_int32 NAPI GA_initFuncs::GetDisplayOutput(void)
```

### Prototype In

snap/graphics.h

### Return Value

Flags of currently active display output device(s).

### Description

This function allows the application to tell what display devices are currently being used to display the image for the user.

## GetMonitorInfo

Returns the currently configured monitor information for the device.

### Declaration

```
void NAPI_GA_initFuncs::GetMonitorInfo(  
    GA_monitor *monitor)
```

### Prototype In

snap/graphics.h

### Parameters

*monitor*                      Place to store the current monitor information

### Description

This function returns a copy of the currently configured monitor for the device.

### See Also

*SetMonitorInfo, GA\_saveMonitorInfo, GA\_detectPnPMonitor*



## GetNumberOfHeads

Return the number of physical output heads supported by the device

### Declaration

```
N_int32 NAPI GA_initFuncs::GetNumberOfHeads(void)
```

### Prototype In

snap/graphics.h

### Return Value

Number of available output heads

### Description

This function is used to determine if the hardware device supports multiple output heads, and if so how many heads. By default all drivers will return a value of 1 if they do not support multiple heads, but devices can support as many heads as the hardware is capable of (to date only dual and triple head hardware exist).

If the hardware has multiple output heads, the *SetActiveHead* and *GetActiveHead* functions can be used to change the active head being used.

### See Also

*SetActiveHead*, *GetActiveHead*

## GetOptions

Returns the current device driver options from the graphics device driver.

### Declaration

```
void NAPI GA_initFuncs::GetOptions(  
    GA_options *options)
```

### Prototype In

snap/graphics.h

### Parameters

*options*                      Place to store the returned options information

### Description

This function returns a structure which contains device driver configuration options for the installed device driver.

---

**Note:** *The dwSize member of the profile structure is intended for future compatibility, and must be set to the size of the structure before calling this function. Only the number of bytes set in the dwSize member will be copied into the callers structure.*

---

### See Also

*SetOptions, GA\_saveOptions*

## GetUniqueFilename

Returns a unique filename for the chipset driver

### Declaration

```
void NAPI_GA_initFuncs::GetUniqueFilename(
    char *filename,
    int type)
```

### Prototype In

snap/graphics.h

### Parameters

<i>filename</i>	Place to store the unique filename
<i>type</i>	Type of of information being stored

### Description

This function generates a unique filename for the current chipset configuration that is used to store POST register information and other configuration options to disk at runtime. Because the filename is unique, each card that is plugged into the system will have a different set of persistent POST register state and configuration values. The type parameter defines the type of information the unique filename will be used for, and presently supports the following:

<i>type</i>	Filename generated
1	modes\%chipname%.%revid%
2	options\%chipname%.%revid%
3	%chipname%.lcd
4	clock.%deviceindex%

This is an internal function and generally should not be called directly by the application program or shell driver.

## GetVideoMode

Returns the currently active display mode number.

### Declaration

```
N_uint32 NAPI GA_initFuncs::GetVideoMode(void)
```

### Prototype In

snap/graphics.h

### Return Value

Currently active display mode.

### Description

This function returns the internal mode number for the currently active display mode. This is mainly useful for finding out what the current display mode is set to when applications are sharing an OS global device driver. Also note that when the SNAP driver is first loaded, it will attempt to interrogate the hardware registers to determine what the current text display mode is, and this function will return the id of the equivalent SNAP display mode.

### See Also

*SetVideoMode, GetVideoModeInfo, GetCustomVideoModeInfo*

## GetVideoModeInfo

Returns information about a specific display mode.

### Declaration

```
N_int32 NAPI GA_initFuncs::GetVideoModeInfo(
    N_uint32 mode,
    GA_modeInfo *modeInfo)
```

### Prototype In

snap/graphics.h

### Parameters

<i>mode</i>	Mode number to get information for
<i>modeInfo</i>	Place to store the returned mode information ( <i>GA_modeInfo</i> )

### Return Value

0 on success, -1 on failure

### Description

This function returns complete information about a specific SNAP display mode from the mode list pointed to by the AvailableModes pointer in the *GA\_devCtx* structure. This function fills the *GA\_modeInfo* structure with detailed information about the requested mode. This function will fail if you pass in a mode number that is not listed in the AvailableModes list.

This function returns the mode information for the currently active output device. To find out information specific to a particular output device (ie: TV, LCD flat panel etc), please use the *GetVideoModeInfoExt* function instead.

---

**Note:** *Modes listed in the AvailableModes list may not be available at runtime, as they may be mapped out based on the maximum pixel clock limits for the display controller, or based on the frequency limits of the attached monitor.*

**Note:** *The calling code must first ensure that the dwSize member of the GA\_modeInfo structure is set to the size of the structure in bytes before calling this function. Only the number of bytes set in the dwSize member will be copied into the callers structure.*

---

### See Also

*GetVideoModeInfoExt, GetCustomVideoModeInfo, GetCustomVideoModeInfoExt, GetCurrentVideoModeInfo, GetCurrentRefreshRate, SetVideoMode*

## GetVideoModeInfoExt

Returns information about a display mode for a specific output device

### Declaration

```
N_int32 NAPI GA_initFuncs::GetVideoModeInfoExt (
    N_uint32 mode,
    GA_modeInfo *modeInfo,
    N_int32 outputDevice)
```

### Prototype In

snap/graphics.h

### Parameters

<i>mode</i>	Mode number to get information for
<i>modeInfo</i>	Place to store the returned mode information ( <i>GA_modeInfo</i> )
<i>outputDevice</i>	Output device flags to use ( <i>GA_OutputFlagsType</i> )

### Return Value

0 on success, -1 on failure

### Description

This function returns extended information about a specific SNAP display mode for a specific output device selection. This is different to the normal *GetVideoModeInfo* function, which returns the display mode information for the currently active output device (ie: LCD, CRT or TV). The only difference is the addition of the *outputDevice* parameter, which contains flags from the *GA\_OutputFlagsType* enumeration.

This function is useful if you are running on a different output device (ie: CRT display) but wish to find out the capabilities of another output device (ie: LCD or TV), without needing to first switch to that device and make it active.

---

**Note:** *The calling code must first ensure that the *dwSize* member of the *GA\_modeInfo* structure is set to the size of the structure in bytes before calling this function. Only the number of bytes set in the *dwSize* member will be copied into the callers structure.*

---

### See Also

*GetVideoModeInfo, GetCustomVideoModeInfo, GetCustomVideoModeInfoExt, GetCurrentVideoModeInfo, GetCurrentRefreshRate, SetVideoMode*

## PerformDisplaySwitch

Performs or fix up the pending display mode switch

### Declaration

```
void NAPI_GA_initFuncs::PerformDisplaySwitch(void)
```

### Prototype In

snap/graphics.h

### Description

This function will perform the requested or pending display display mode switch. This will be done in a non-destructive manner such that the screen and current hardware acceleration state are preserved. This function is used to dynamically switch the active display device and re-program the hardware as necessary for the new active display device (ie: LCD/CRT/TV switching etc).

---

**Note:** *Make sure you also call PostSwitchPhysicalResolution after calling this function!*

---

### See Also

*PollForDisplaySwitch, SwitchPhysicalResolution, PostSwitchPhysicalResolution*

## PollForDisplaySwitch

Polls the hardware to determine if a mode change is pending

### Declaration

```
N_int32 NAPI GA_initFuncs::PollForDisplaySwitch(void)
```

### Prototype In

snap/graphics.h

### Return Value

True if a mode change occurred, false if not.

### Description

This function polls the hardware to determine if a mode change should be processed, and if so it processes the mode change and returns to the caller. This function should be called about once every 250ms by the OS shell display driver to ensure that mode changes are handled in a timely manner. Mode changes include switching the active display device via system hot keys (ie: LCD to CRT to TV etc), as well as changing the expansion settings on laptop systems etc. Internally SNAP drivers will use this function to determine if a switch is pending or if a switch has already taken place that needs to be 'fixed up'. If a switch is pending, the OS shell driver should then call the *PerformDisplaySwitch* function to complete the process of performing the requested display switch.

### See Also

*PerformDisplaySwitch*, *SwitchPhysicalResolution*



## SaveCRTCTimings

Sets the current CRTC timings for the active display mode

### Declaration

```
void NAPI_GA_initFuncs::SaveCRTCTimings(
    GA_CRTCInfo *crtc)
```

### Prototype In

snap/graphics.h

### Parameters

*crtc*                      New CRTC timings to program and make the default

### Description

This function sets the active CRTC timings for the active display mode in the device driver, and also makes those timings the default timings for that display mode. After this function is called, the hardware will have been re-programmed for the new CRTC timings, and the internal database of CRTC timings will have been updated to reflect these new timings. The CRTC timings will only be stored in memory, but they can be made permanent with a call to the *GA\_saveCRTCTimings* function which will flush the changes to the disk based copy of the CRTC database.

### See Also

*GetCRTCTimings, SetCRTCTimings, GetCurrentRefreshRate, SetGlobalRefresh, GA\_saveCRTCTimings, GA\_restoreCRTCTimings, GA\_getCRTCTimings, GA\_setCRTCTimings, GA\_setDefaultRefresh*

## SaveRestoreState

Save or restore the state of the display hardware.

### Declaration

```
N_int32 NAPI GA_initFuncs::SaveRestoreState(
    N_int32 subfunc,
    void *saveBuf)
```

### Prototype In

snap/graphics.h

### Parameters

<i>subfunc</i>	Sub-function number
<i>saveBuf</i>	Buffer to save or restore data from.

### Return Value

Size of the save buffer if subfunc is set to 2.

### Description

This function provides support for saving and restoring the complete hardware state. This is useful for debuggers and other utility software that needs to be able to temporarily take over the display and restore it back to the original state it was in. This function is responsible for saving and restoring all hardware registers related to the graphics mode.

A buffer large enough to hold the entire hardware state must be allocated by the calling code and passed in the saveBuf parameter. In order to determine the size of the hardware state buffer to be allocated, the calling code should first call this function with subfunc set to 2 to determine the size of the hardware save/restore state buffer.

The following subfunctions are defined:

<i>subfunction</i>	description
<b>0</b>	Save hardware state to saveBuf
<b>1</b>	Restore hardware state from saveBuf
<b>2</b>	Return state buffer size

---

**Note:** *This function is not yet implemented in the SNAP 1.0 spec, and it may be obsoleted in a future specification.*

---

## SetActiveHead

Set the currently active output head for the device

### Declaration

```
N_int32 NAPI GA_initFuncs::SetActiveHead(  
    N_int32 headIndex)
```

### Prototype In

snap/graphics.h

### Parameters

*headIndex*                      Index of head to make active (*GA\_multiHeadType*)

### Return Value

Index of previously active head (*GA\_multiHeadType*)

### Description

This function is used to change the currently active output head for the device. Once the active head is changed, all calls to *GetVideoModeInfo* and *SetVideoMode* will be specific to the newly active head.

Note that by default device drivers that support multiple heads will usually default to *gaActiveHeadClone* mode, such that the same information will be display on all heads initially.

### See Also

*GetNumberOfHeads*, *GetActiveHead*

## SetCRTCTimings

Sets the currently active CRTC timings for the active display mode.

### Declaration

```
void NAPI_GA_initFuncs::SetCRTCTimings(  
    GA_CRTCInfo *crtc)
```

### Prototype In

snap/graphics.h

### Parameters

*crtc*                      Set CRTC timings to make active

### Description

This function sets the active CRTC timings for the active display mode in the device driver. After this function is called, the hardware will have been re-programmed for the new CRTC timings, but the internal CRTC database will *not* have been updated. Hence the changes are temporary and the next time a display mode is set the values will be restored to the previous settings.

### See Also

*SaveCRTCTimings, GetCRTCTimings*

## SetCustomVideoMode

Sets a specified custom display mode.

### Declaration

```
N_int32 NAPI GA_initFuncs::SetCustomVideoMode(
    N_int32 xRes,
    N_int32 yRes,
    N_int32 bitsPerPixel,
    N_uint32 flags,
    N_int32 *virtualX,
    N_int32 *virtualY,
    N_int32 *bytesPerLine,
    N_int32 *maxMem,
    GA_CRTCInfo *crtc)
```

### Prototype In

snap/graphics.h

### Parameters

<i>xRes</i>	Physical X resolution for the display mode
<i>yRes</i>	Physical Y resolution for the display mode
<i>bitsPerPixel</i>	Color depth for the display mode
<i>flags</i>	Mode initialisation flags
<i>virtualX</i>	Requested virtual display X resolution (-1 to use default)
<i>virtualY</i>	Requested virtual display Y resolution (-1 to use default)
<i>bytesPerLine</i>	Returns the scanline width for the mode
<i>maxMem</i>	Returns the maximum addressable display memory limit
<i>crtc</i>	CRTC information block (required)

### Return Value

0 on success, -1 on failure

### Description

This function is used to initialize a specific custom display mode. The custom display mode does not have to be one of that matches the resolution and refresh rate for modes stored in the AvailableModes list of the *GA\_devCtx* structure. Any value outside what the hardware is capable of displaying, will cause this function to return a failure condition. Make sure you first call *GetCustomVideoModeInfo* to determine if the requested mode is actually supported.

For the most part this function is identical to the regular *SetVideoMode* function, but works with custom display modes rather than the list of valid modes in the device driver mode profile. Since internally SNAP drivers have no concept of display modes, the *SetVideoMode* function internally ends up calling this function to actually initialise a display mode.

This function also accepts the flags defined in *GA\_modeFlagsType* enumeration passed in the flags parameter. These flags change the way that the selected display mode is initialized, and are identical to the flags passed to the regular *SetVideoMode* function.

---

**Note:** This function ***requires*** a set of CRTC parameters to be passed in the *crtc* parameter, unlike *SetVideoMode*.

---

**See Also**

*SetVideoMode*, *GetVideoModeInfo*, *GetCustomVideoModeInfo*, *SetDisplayOutput*,  
*GetClosestPixelClock*

## SetDisplayOutput

Sets the output device(s) for the display

### Declaration

```
N_int32 NAPI GA_initFuncs::SetDisplayOutput(
    N_int32 device)
```

### Parameters

*device*                      Device flags for the displays to make active  
                                  (*GA\_OutputFlagsType*)

### Return Value

Flags of previously active devices or -1 on error.

### Description

This function allows the application to switch the currently active display between the CRT, LCD or TV etc. Check that the mode information indicates that LCD display or TVOut display is supported before trying to switch to that display mode or this function will fail. You can also combine the flags together so that *SetDisplayOutput(gaOUTPUT\_CRT | gaOUTPUT\_LCD)* will switch to simultaneous display on both the LCD and CRT display (assuming the device supports this mode of operation; if not it will fail). Likewise *SetDisplayOutput(gaOUTPUT\_CRT | gaOUTPUT\_TV)* will display on both the CRT display and TVOut connector. Please check the *GA\_OutputFlagsType* for the list of valid identifiers that can be passed to this function.

## SetGlobalRefresh

Sets the global default refresh rate for all high resolution display modes.

### Declaration

```
void NAPI GA_initFuncs::SetGlobalRefresh(
    N_int32 refresh)
```

### Prototype In

snap/graphics.h

### Parameters

*refresh*                      New refresh rate to make the global default

### Description

This function will set the default refresh rate for all high resolution display modes (all modes  $\geq 640 \times 480$ ) to the specified refresh rate. If the hardware cannot achieve the specified refresh rate, the next lowest refresh available will be made the default instead. The changes are only made to the in memory copy of the CRTC database, but they can be made permanent with a call to the *GA\_saveCRTCTimings* function.

### See Also

*SetCRTCTimings*, *GetCRTCTimings*, *SetCRTCTimings*, *GetCurrentRefreshRate*,  
*GA\_saveCRTCTimings*, *GA\_restoreCRTCTimings*, *GA\_getCRTCTimings*,  
*GA\_setCRTCTimings*, *GA\_setDefaultRefresh*



## SetModeProfile

Sets the current mode profile for the graphics device driver.

### Declaration

```
void NAPI GA_initFuncs::SetModeProfile(
    GA_modeProfile *profile)
```

### Prototype In

snap/graphics.h

### Parameters

*profile*                      Mode profile to make active for the display driver.

### Description

This function installs a new mode profile into the driver to be used as the default mode profile. The mode profile is a structure which contains configuration information about the available display resolutions for the installed device driver. A default mode profile is shipped with the graphics device drivers, but a new mode profile can be downloaded at any time. Since SNAP drivers internally do not have any concept of specific display resolutions, the mode profile is used to tell the SNAP driver what specific display resolutions the driver should export to user applications. Hence changing the mode profile can be used to add new display modes, or remove unwanted display modes from the list of display modes supported by the driver.

The mode profile may be made permanent with a call to the *GA\_saveModeProfile* function, to be used every time a SNAP driver is loaded from disk.

---

**Note:** *All mode in the mode profile must also have the associated CRTC tables added to the CRTC database before it will work (created using GTF if the mode is a new custom display mode).*

**Note:** *The dwSize member of the profile structure is intended for future compatibility, and must be set to the size of the structure before calling this function. Only the number of bytes set in the dwSize member will be copied from the callers structure.*

---

### See Also

*GetModeProfile, GetVideoModeInfo, GA\_saveModeProfile*

## SetMonitorInfo

Sets the currently configured monitor information for the device.

### Declaration

```
void NAPI_GA_initFuncs::SetMonitorInfo(  
    GA_monitor *monitor)
```

### Prototype In

snap/graphics.h

### Parameters

*monitor*                      New monitor information to make active

### Description

This function sets the currently configured monitor for the device. Once this function has been called, all SNAP display modes will be filtered based on the capabilities determined by the monitor information that was made active. Note that this function only updates the in memory copy of the active monitor. To make this permanent, call *GA\_saveMonitorInfo* which will save the monitor record to disk.

### See Also

*GetMonitorInfo*, *GA\_saveMonitorInfo*, *GA\_detectPnPMonitor*

## SetOptions

Sets the device driver options for the graphics device driver.

### Declaration

```
void NAPI_GA_initFuncs::SetOptions(  
    GA_options *options)
```

### Prototype In

snap/graphics.h

### Parameters

*options*                      Device driver options to make active for the display driver.

### Description

This function installs a new set of device driver options that control the operation of the device driver at runtime. A default set of options is always built into the device driver, but the options can be changed at any time. The options may be made permanent with a call to the *GA\_saveOptions* function.

---

**Note:** *The dwSize member of the profile structure is intended for future compatibility, and must be set to the size of the structure before calling this function. Only the number of bytes set in the dwSize member will be copied from the callers structure.*

---

### See Also

*GetOptions, GA\_saveOptions*

## SetRef2dPointer

Pass a pointer to the ref2d structure to the device driver

### Declaration

```
void NAPI_GA_initFuncs::SetRef2dPointer(  
    struct _REF2D_driver *ref2d)
```

### Prototype In

snap/graphics.h

### Description

This is an internal function that is used by the 2d reference rasteriser library to register itself with the hardware driver, so the hardware driver can call back into the ref2d driver to access it's internal functions. This is mostly used by drivers implementing hardware video acceleration functions, so they driver can call the ref2d buffer manager to allocate and manager offscreen video memory buffers.

SEE ALSO *SetSoftwareRenderFuncs*

## SetSoftwareRenderFuncs

Sets the software rendering callback vectors in the hardware driver

### Declaration

```
void NAPI GA_initFuncs::SetSoftwareRenderFuncs(  
    GA_2DRenderFuncs *softwareFuncs)
```

### Prototype In

snap/graphics.h

### Description

This is an internal function that is used by the 2d reference rasteriser library to register all the current software rendering functions with the hardware driver. This is done after the software rendering functions are loaded and initialised by the application or shell driver. These functions are used internally by the SNAP device driver to fall back to software rendering to handle special cases where the hardware cannot implement certain features or to work around hardware bugs.

This function should generally not be called directly by application programs (unless you have completely replaced the 2d reference rasteriser component).

SEE ALSO *SetRef2dPointer*

## SetVideoMode

Sets a specified display mode.

### Declaration

```
N_int32 NAPI GA_initFuncs::SetVideoMode(
    N_uint32 mode,
    N_int32 *virtualX,
    N_int32 *virtualY,
    N_int32 *bytesPerLine,
    N_int32 *maxMem,
    N_int32 refreshRate,
    GA_CRTCInfo *crtc)
```

### Prototype In

snap/graphics.h

### Parameters

<i>mode</i>	Mode number to set, including flags
<i>virtualX</i>	Requested virtual display X resolution (-1 to use default)
<i>virtualY</i>	Requested virtual display Y resolution (-1 to use default)
<i>bytesPerLine</i>	Returns the scanline width for the mode
<i>maxMem</i>	Returns the maximum addressable display memory limit
<i>refreshRate</i>	Refresh rate to set (0 for default)
<i>crtc</i>	CRTC information block if afRefreshCtrl specified

### Return Value

0 on success, -1 on failure

### Description

This function is used to initialize a specified display mode. The mode number passed to this function should be one of the values stored in the AvailableModes list of the *GA\_devCtx* structure combined with any of the valid mode set flags defined in the *GA\_modeFlagsType* enumeration. Any value outside of this set of values will cause this function to fail.

When the graphics mode is initialized, you can pass in a specific virtual X resolution to enable a wide virtual display mode, or an interleaved stereo display mode. If you pass in a value of -1, the physical X resolution for the display mode will be used to initialise the virtual X resolution. If you pass in a value other than -1, the driver will attempt to satisfy your request with the next largest value that the controller can actually support. The value programmed will be returned in the virtualX parameter on exit. It is possible that the graphics mode cannot have the scanline width changed, or that the scanline width requested was too large for the graphics mode to handle, in which case this function will fail.

The virtualY parameter should be filled in with the virtual desktop height if you plan to do virtual panning, and is mostly used to compute the number of available display pages for page flipping etc, as well as where the start of offscreen memory is for the

offscreen buffer manager. If this value is set to -1, the physical Y resolution of the display mode will be used to initialise the virtual Y resolution. The actual value used will be returned in the virtualY parameter on exit.

On exit this function returns the number of bytes in a logical scanline in the bytesPerLine parameter. This can then be used to implement software rendering directly to the hardware linear framebuffer. The maximum addressable display memory limit is returned in the 'maxMem' parameter and defines the byte offset of the highest linear framebuffer address that can be used by the application. The amount of addressable framebuffer display memory may not extend to the end of physical display memory due to memory used by the driver for hardware cursors and patterns etc.

You can also elect to either use the default refresh rate for the display mode or force a specific refresh rate with the refreshRate parameter. If a value of 0 is passed for the refreshRate parameter, the user selected default refresh rate will be used. A value other than zero will try to set the requested refresh rate from the table of available refresh rates that the mode supports. Make sure you first check the list of available refresh rates reported by the *GetVideoModeInfo* function before calling this function, or this function will fail. For generic GTF refresh rate control use the gaRefreshCtrl flag and the refreshRate parameter will be ignored. Instead the CRTC timings passed in the crtc parameter will be used for the display mode. This is most useful for unusual refresh rates not listed in the refresh rate list, such as 120Hz and higher refresh rates of LC stereo shutter glasses, or to hit exact CRTC timings for special flat panel or fixed frequency monitors.

---

**Note:** *It is highly recommended that applications do not change the user selected default refresh rate except for special circumstances where a specific refresh rate is required (such as 120Hz stereo for instance).*

---

This function also accepts the flags defined in *GA\_modeFlagsType* enumeration logically OR'ed with the passed display mode number. These flags change the way that the selected display mode is initialized as follows:

The gaDontClear flag is used to specify that the video memory should not be cleared when the graphics mode is initialized. By default the video memory will be cleared to all 0's by the driver.

The gaLinearBuffer flag is used to specify that the application wishes to enable the linear framebuffer version of the graphics mode. On many controllers, the acceleration features are only available in the linear framebuffers modes. Also note that on many new PCI devices, PCI burst mode is only enabled in the linear framebuffer modes, so these modes should be used whenever possible for maximum performance. Make sure that you check the gaHaveLinearBuffer flag in the Attributes member of the *GA\_modeInfo* structure to determine if this is supported in the selected display mode.

The ga6BitDAC flag is used to force 8-bit display modes to set the RAMDAC to a VGA compatible 6-bit per primary mode, rather than the default 8-bit per primary mode. This is generally used to make the display mode compatible with the default VESA VBE and

VGA display modes. Normally applications will not want to set this but instead leave the mode in the default 8-bits per primary setting. Note also that this has no effect on 15-bpp and higher display modes.

The `gaNoAccel` flag is used to initialise the display mode without initialising the hardware acceleration portions of the graphics chipset. This is generally used to implement VESA VBE compatible display modes where the hardware accelerator functions are not used (and the hardware accelerator state will not be touched or changed).

The `gaRefreshCtrl` flag can be set to enable generic GTF refresh rate control for the display mode. If this bit is set, the mode will be using the CRTC parameters and pixel clock values passed in the `crtc` parameter, rather than using the value passed in the `refreshRate` parameter. This allows the application program or operating system drivers to calculate a new set of CRTC values (preferably using the VESA Generalized Timing Formula, or GTF specification) for the mode, and allow the refresh rate to be set to any supported value for the hardware.

The `gaWindowedMode` flag is specific to emulation display drivers such as the DirectX display driver. These drivers are implemented on top of an existing abstraction layer, and this flag is used to inform the driver that the mode being set is actually a windowed mode (ie: switch back to the regular desktop display mode).

The `gaPartialModeSet` flag can be set to tell the SNAP driver to only perform a partial mode set instead of a complete mode set. When this flag is used, the SNAP driver will only re-program the CRTC controller and pixel clock, but will not re-program the graphics engine components. This flag is primarily used to enable SNAP drivers to change the CRTC timings and refresh rate for an existing display driver without destroying the remaining state of the graphics card.

---

**Note:** *When a linear framebuffer mode is enabled, it is the responsibility of the driver to ensure that all VGA memory resources such as the 0xA0000-0xBFFFF regions are disabled if possible. Ensuring these regions are disabled provides for the maximum performance when multiple display controllers are present in the system.*

---

#### See Also

`SetCustomVideoMode`, `GetVideoModeInfo`, `GetCustomVideoModeInfo`, `SetDisplayOutput`, `GetClosestPixelClock`



## SwitchPhysicalResolution

Switches the physical resolution and refresh rate for a display mode

### Declaration

```
N_int32 NAPI GA_initFuncs::SwitchPhysicalResolution(
    N_int32 physicalXResolution,
    N_int32 physicalYResolution,
    N_int32 refreshRate)
```

### Prototype In

snap/graphics.h

### Parameters

<i>physicalXResolution</i>	New physical X resolution to program
<i>physicalYResolution</i>	New physical Y resolution to program
<i>refreshRate</i>	New refresh rate to program

### Return Value

0 on success, -1 on failure

### Description

This function is used to switch the physical resolution or refresh rate for the current display mode. This will be done in a non-destructive manner such that the screen and current hardware acceleration state are preserved. This function is used primarily to enable and disable different levels of hardware panning or 'zooming' in the display driver (ie: 640x480 physical display resolution, 1024x768 virtual). It can also be used to change the refresh rate without changing the physical resolution.

If the *physicalXResolution* or *physicalYResolution* parameters are set to -1, the physical resolution of the screen is obtained from the physical resolution of the currently active display mode, otherwise the passed in values are used. The refresh rate is always used, but if you pass in a value of 0, the default refresh rate for the current display mode will be used to program the hardware.

### See Also

*PollForDisplaySwitch*, *PerformDisplaySwitch*, *SwitchPhysicalResolution*

## GA\_largeInteger

### Declaration

```
typedef u64 GA_largeInteger
```

### Prototype In

snap/graphics.h

### Description

Defines the structure for holding 64-bit integers used for storing the values returned by the precision timing functions. The precision timing functions are used internally by the the SNAP Graphics drivers for software stereo support, however the granularity of the timing functions is variable. Generally a granularity of around 1us is desired for maximum accuracy. Where possible these import functions should be implemented using the Intel Pentium RDTSC instruction or equivalent (with time readings normalised to 1us granularity to avoid overflow internally).

### Members

<i>low</i>	Low 32-bits of the 64-bit integer
<i>high</i>	High 32-bits of the 64-bit integer

## GA\_layout

### Declaration

```
typedef struct {  
    N_uint32    left;  
    N_uint32    top;  
    N_uint32    right;  
    N_uint32    bottom;  
} GA_layout
```

### Prototype In

snap/graphics.h

### Description

Structure used to determine the layout in multi-controller modes. The layout is defined as adjacent rectangles for each device, and should be set up using the multi-controller setup program.

### Members

<i>left</i>	Left coordinate for layout rectangle
<i>top</i>	Top coordinate for layout rectangle
<i>right</i>	Right coordinate for layout rectangle
<i>bottom</i>	Bottom coordinate for layout rectangle

## GA\_loaderFuncs

**Prototype In**

snap/graphics.h

**Description**

Function group containing all internal loading and unloading functions for the device. These functions are used internally by the SNAP device driver loading mechanism and should *never* be called directly by application or shell driver code.

## InitDriver

Initialises the driver for operation

### Declaration

```
N_int32 NAPI GA_loaderFuncs::InitDriver(void)
```

### Prototype In

snap/graphics.h

### Return Value

nOK on success, SNAP error code if not (see *N\_errorType*)

### Description

This function is called as part of the internal initialisation process for the SNAP driver, so application code should *never* call this function directly.

### See Also

*InitDriver*, *QueryFunctions*, *UnloadDriver*

## QueryFunctions

Returns the function pointers for the specified function group.

### Declaration

```
ibool NAPI GA_loaderFuncs::QueryFunctions(
    N_uint32 id,
    N_int32 safetyLevel,
    void _FAR_ *funcs)
```

### Prototype In

snap/graphics.h

### Parameters

<i>id</i>	Identifier for the function group to get pointers for
<i>safetyLevel</i>	Safety level requested
<i>funcs</i>	Pointer to function block to fill in

### Return Value

True if the requested function group is available, false if not.

### Description

This function is the main function that is used by the graphics application code to get a block of function pointers for a specified function group. The function groups are defined *GA\_funcGroupsType* enumeration, and breaks up the functions in the device driver API into groups of logically similar functions. For instance to get the block of functions for the hardware cursor, you would call this function with the *GA\_GET\_CURSORFUNCS* identifier.

The *safetyLevel* parameter defines the safety level of those functions that are returned by the driver. The safety level essentially defines what runtime requirements those functions have, such as requiring kernel mode access to registers etc. There are currently three safety levels defined:

<i>safetyLevel</i>	Description
0	Level 0 is defined as 'unsafe', which means that the functions must be executed in kernel mode as they require access to I/O space registers. Essentially requesting level 0 function pointers will return pointers to all available functions in that function group.
1	Level 1 is defined as 'mostly safe', which means that the functions only access memory mapped registers and do not require any I/O space register access. These functions can thus be executed in user space provided that all the memory mapped register regions are mapped into user space.
2	Level 2 is defined as 'safe', which means that the functions only access memory mapped registers via the 'safe'

IOMemMaps[2] and IOMemMaps[3] regions (the first two will only be mapped into kernel space by the loader code). The idea here is that the driver separates 'safe' and 'unsafe' registers (defined below).

Note that with the above defined safety levels, it is possible for the display driver in the operating system to provide a compromise between maximum performance and maximum security as a user level setting. Maximum security would be achieved if the graphics functions were only ever called from kernel mode, but would incur the overhead of user mode to kernel mode switching for all graphics output. Better performance can be achieved if safety level 2 functions are executed in the user mode component of the display driver, but this requires specific hardware that is designed to separate 'safe' and 'unsafe' registers by at least 8Kb (so they can be individually mapped into user+kernel or just kernel space). Even higher performance can be achieved if all functions which only use memory mapped register access are executed in user space, which can be done with all modern PC graphics hardware. However this does expose the possibility of an errant application accidentally writing to the user space registers and causing the graphics hardware to lock.

---

**Note:** *Application code should not call this function directly, but instead call GA\_queryFunctions.*

**Note:** *To allow for future compatibility, all function blocks begin with a dwSize member. The caller is expected to fill in the dwSize member with the size of the function block being retrieved before calling QueryFunctions. If the driver exports more functions than the application knows about, only a subset of the functions are copied to the application. If the application expects more functions than the driver provides, the non-existent functions are set to NULL pointers by QueryFunctions, and the remainder copies from the driver.*

**Note:** *The OS loader code will only map the IOMemMaps[2] and IOMemMaps[3] regions into user space, and will map all memory mapped I/O regions into kernel space. This allows unsafe registers (that would potentially cause the system to lock) to not be mapped into user space and potentially compromising system stability.*

**Note:** *This mechanism also provides for a clean and simple upgrade path for future drivers, while ensuring maximum compatibility with existing specifications. New functions for a particular group can simply be added to the end of the function group to extend that group. Totally new function groups can be added by defining new identifiers for that function group, and older drivers will return a NULL if that function group is requested. Finally if a function group requires a complete redesign to achieve maximum performance for next generation hardware, a new extended function group can be defined (and drivers can continue to export the older and slower function group for backwards compatibility).*

---

**Note:** *Safety levels other than 0 are not implemented in SNAP 1.0*

---

#### **See Also**

*InitDriver, QueryFunctions, UnloadDriver*

## UnloadDriver

Prepare the device driver for unloading

### Declaration

```
void NAPI_GA_loaderFuncs::UnloadDriver(void)
```

### Prototype In

snap/graphics.h

### Description

This function is called as part of the internal device driver unloading process. This function is called by the loader code when a device driver is about to be unloaded, giving the device driver a chance to free up any internal memory allocations ready to be unloaded completely.

Application code should *never* call this function directly.

### See Also

*InitDriver, QueryFunctions, UnloadDriver*



## GA\_mixCodesType

### Declaration

```
typedef enum {
    GA_R2_BLACK,
    GA_R2_NOTMERGESRC,
    GA_R2_MASKNOTSRC,
    GA_R2_NOTCOPYSRC,
    GA_R2_MASKSRCNOT,
    GA_R2_NOT,
    GA_R2_XORSRC,
    GA_R2_NOTMASKSRC,
    GA_R2_MASKSRC,
    GA_R2_NOTXORSRC,
    GA_R2_NOP,
    GA_R2_MERGENOTSRC,
    GA_R2_COPYSRC,
    GA_R2_MERGESRCNOT,
    GA_R2_MERGESRC,
    GA_R2_WHITE,
    GA_REPLACE_MIX = GA_R2_COPYSRC,
    GA_AND_MIX     = GA_R2_MASKSRC,
    GA_OR_MIX      = GA_R2_MERGESRC,
    GA_XOR_MIX     = GA_R2_XORSRC,
    GA_NOP_MIX     = GA_R2_NOP
} GA_mixCodesType
```

### Prototype In

snap/graphics.h

### Description

Logical mix operation codes for accelerated rendering functions that support mixes. The set of mix codes is the standard Microsoft Raster Operation (ROP2) codes between two values. We define our ROP2 codes as being between the source and destination pixels for blt's, between the foreground or background color and the destination pixels for solid and mono pattern fills and between the pattern pixels and the destination pixels for color pattern fills. It is up to the driver to do any necessary translation between these generic ROP2 codes and each different type of hardware mix code internally. Next to each code is the equivalent Microsoft defined ROP3 code between source and destination pixels.

---

**Note:** *Some graphics controllers may not support all mix codes, so you must use the GetMixTable function to determine the set of mix codes that the controller supports. Setting a mix code that is not listed in the returned mix table will result in undefined behaviour.*

---

### Members

GA_R2_BLACK	0
GA_R2_NOTMERGESRC	DSon
GA_R2_MASKNOTSRC	DSna
GA_R2_NOTCOPYSRC	Sn
GA_R2_MASKSRCNOT	SDna
GA_R2_NOT	Dn

<i>GA_R2_XORSRC</i>	DSx
<i>GA_R2_NOTMASKSRC</i>	DSan
<i>GA_R2_MASKSRC</i>	DSa
<i>GA_R2_NOTXORSRC</i>	DSxn
<i>GA_R2_NOP</i>	D
<i>GA_R2_MERGENOTSRC</i>	DSno
<i>GA_R2_COPYSRC</i>	S
<i>GA_R2_MERGESRCNOT</i>	SDno
<i>GA_R2_MERGESRC</i>	DSo
<i>GA_R2_WHITE</i>	1
<i>GA_REPLACE_MIX</i>	Replace mode
<i>GA_AND_MIX</i>	AND mode
<i>GA_OR_MIX</i>	OR mode
<i>GA_XOR_MIX</i>	XOR mode
<i>GA_NOP_MIX</i>	Destination pixel unchanged

## GA\_mode

### Declaration

```
typedef struct {  
    short    xRes;  
    short    yRes;  
    uchar    bits;  
} GA_mode
```

### Prototype In

snap/graphics.h

### Description

Structure used to describe the available display modes in the SNAP Graphics options structure. This allows the end user to add and delete available display modes from the SNAP Graphics drivers easily using our generic SNAP Graphics driver interface.

### Members

<i>xRes</i>	Horizontal pixel resolution
<i>yRes</i>	Vertical pixel resolution
<i>bits</i>	Color depth per pixel (0 = text mode)

## GA\_modeFlagsType

### Declaration

```
typedef enum {
    gaDontClear                = 0x8000,
    gaLinearBuffer             = 0x4000,
    ga6BitDAC                  = 0x2000,
    gaNoAccel                  = 0x1000,
    gaRefreshCtrl              = 0x0800,
    gaWindowedMode             = 0x0400,
    gaPartialModeSet           = 0x0200,
    gaModeMask                  = 0x01FF
} GA_modeFlagsType
```

### Prototype In

snap/graphics.h

### Description

This enumeration defines the flags for combining with graphics mode numbers to be passed to the SetVideoMode function.

### Members

<i>gaDontClear</i>	Don't clear display memory
<i>gaLinearBuffer</i>	Enable linear framebuffer mode
<i>ga6BitDAC</i>	Set the mode with a 6-bit RAMDAC instead of 8
<i>gaNoAccel</i>	Set the mode without any acceleration support
<i>gaRefreshCtrl</i>	Enable refresh rate control
<i>gaWindowedMode</i>	Initialise for use in the current desktop mode
<i>gaPartialModeSet</i>	Initialise the driver internals, but don't program hardware
<i>gaModeMask</i>	Mask to remove flags and extract VBE mode number

## GA\_modeInfo

### Declaration

```
typedef struct {
    N_uint32      dwSize;
    N_uint32      Attributes;
    N_uint16      XResolution;
    N_uint16      YResolution;
    N_uint8       XCharSize;
    N_uint8       YCharSize;
    N_uint32      BytesPerScanLine;
    N_uint32      MaxBytesPerScanLine;
    N_uint32      MaxScanLineWidth;
    N_uint32      MaxScanLines;
    N_uint32      LinearHeapStart;
    N_uint32      MaxLinearOffset;
    N_uint16      BitsPerPixel;
    GA_pixelFormat PixelFormat;
    N_uint16      MaxBuffers;
    N_uint32      MaxPixelClock;
    N_int32       DefaultRefreshRate;
    N_int32       _FAR_ *RefreshRateList;
    N_uint32      BitmapStartAlign;
    N_uint32      BitmapStridePad;
    N_uint32      MonoBitmapStartAlign;
    N_uint32      MonoBitmapStridePad;
    GA_bltFx      _FAR_ *BitBltCaps;
    GA_videoInf   _FAR_ * _FAR_ *VideoWindows;
    struct _GA_3DState _FAR_ *HW3DCaps;
    N_float32     MaxOOZ;
    N_float32     MaxOOW;
    N_float32     MaxOOS;
    N_float32     MaxOOT;
    N_uint32      DepthFormats;
    N_uint32      DepthStartAlign;
    N_uint32      DepthStridePad;
    N_uint32      TextureFormats;
    N_uint32      TextureStartAlign;
    N_uint32      TextureStridePad;
    N_uint32      TextureMaxX;
    N_uint32      TextureMaxY;
    N_uint16      TextureMaxAspect;
    N_uint32      StencilFormats;
    N_uint32      StencilStartAlign;
    N_uint32      StencilStridePad;
    N_uint32      LinearSize;
    N_uint32      LinearBasePtr;
    N_uint32      AttributesExt;
    N_uint16      PhysicalXResolution;
    N_uint16      PhysicalYResolution;
} GA_modeInfo
```

### Prototype In

snap/graphics.h

### Description

Graphics mode information block. This structure contains detailed information about the capabilities and layout of a specific graphic mode.

The `Attributes` member contains a number of flags that describes certain important characteristics of the graphics mode, and the values this member contains are defined in the *GA\_AttributeFlagsType*.

The `XResolution` and `YResolution` specify the logical width and height in pixel elements for this display mode. The logical resolution is the resolution of all available pixels in the display which may be larger than the physical resolution if the mode has hardware panning enabled. Hardware panning is enabled if the maximum physical resolution of the display device does not support the specific mode, such as when hot switching between a CRT monitor and an LCD panel or TV output device.

The `BytesPerScanLine` member specifies how many full bytes are in each logical scanline. The logical scanline could be equal to or larger than the displayed scanline, and can be changed when the display mode is first initialized.

The `MaxBytesPerScanLine` and `MaxScanLineWidth` members define the maximum virtual framebuffer coordinates that can be initialised for the mode, in both bytes and pixels respectively. If an attempt is made to initialize a graphics mode with values larger than these values, the mode set will fail.

The `MaxScanLines` member holds the total number of scanlines available in that graphics mode when initialised with the default scanline width. This field combined with `BytesPerScanLine` can be used to determine the maximum addressable display memory for drawing operations. This can also be used to determine how large a virtual screen image can be before initialising a graphics mode. This field also determines the addressable limit for X/Y based drawing functions in offscreen video memory.

The `LinearHeapStart` member determines the start of the linear only heap, if one is available. Some hardware has restrictions on the addressable memory for the (x,y) coordinates passed to the 2D drawing functions. If the hardware supports `DrawRectLin` and the `BitBltLin` family of functions, the memory past the (x,y) coordinate restriction can be accessed using those functions. Hence this member determines the start of this linear only heap as a byte offset from the beginning of display memory. Memory in the linear only heap can only be accessed directly via the linear framebuffer, or using `DrawRectLin` or the `BitBltLin` family of blitting functions. None of the X/Y based drawing functions can be used to draw to the linear only heap.

The `MaxLinearOffset` member hold the maximum addressable display memory offset for linear drawing functions (`DrawRectLin`, `BitBltLin` etc). If the hardware has restrictions on the addressable memory for the X/Y drawing functions, the linear only heap resides between `LinearHeapStart` and `MaxLinearOffset`. If the maximum addressable scanline value for a display mode is past the end of display memory (common for high resolution modes), then `LinearHeapStart` = `MaxLinearOffset` which indicates that there is no linear only heap for that display mode.

---

**Note:** *There may well be some memory used by the graphics hardware for caching the hardware cursor, patterns and other data between the end of (MaxScanLines \* BytesPerScanLine) and LinearHeapStart. Hence the application software must never write to the memory between (MaxScanLines \* BytesPerScanLine) and LinearHeapStart.*

---

The BitsPerPixel member specifies the number of bits per pixel for this display mode. For 5:5:5 format RGB modes this should contain a value of 15, and for 5:6:5 format RGB modes this should contain a value of 16. For 8:8:8 bit RGB modes this should contain a value of 24 and for 8:8:8:8 RGBA modes this should contain a value of 32. For 24 and 32bpp modes, the application should look at the pixel format mask values (see below) to determine the actual format of the pixels within the display buffer.

The MaxBuffers member specified the maximum number of display buffers that can be allocated in video memory for page flipping. This value is a convenience function only, and can be computed manually from the value of MaxScanLines / YResolution.

The MaxPixelClock member specifies the maximum possible pixel dot clock that can be selected in this display mode when a refresh controlled mode is selected. Any attempt to select a higher pixel clock will cause the mode set to fail. This member can be used to determine what the maximum available refresh rate for the display mode will be.

The RedMaskSize, GreenMaskSize, BlueMaskSize and RsvdMaskSize members define the size, in bits, of the red, green, and blue components of an RGB pixel respectively. A bit mask can be constructed from the MaskSize members using simple shift arithmetic. For example, the MaskSize values for an RGB 5:6:5 mode would be 5, 6, 5, and 0, for the red, green, blue, and reserved members respectively.

The RedFieldPosition, GreenFieldPosition, BlueFieldPosition and RsvdFieldPosition members define the bit position within the RGB pixel of the least significant bit of the respective color component. A color value can be aligned with its pixel member by shifting the value left by the FieldPosition. For example, the FieldPosition values for an RGB 5:6:5 mode would be 11, 5, 0, and 0, for the red, green, blue, and reserved members respectively.

The BitmapStartAlign member defines the alignment requirements in bytes for offscreen memory bitmaps for this graphics mode. If the value in here is set to 8 for instance, then the start for all offscreen bitmaps must be aligned to an 8 byte boundary in order to be used for offscreen bitmap blitting. Note that the BitmapStartAlign member also defines the alignment requirements for all buffers passed to the SetDrawBuffer function.

The BitmapStridePad member defines the alignment requirements in bytes for the stride of offscreen memory bitmaps (the number of bytes to move from one line of the bitmap to the next). If the value in here is set to 8 for instance, then the number of bytes for each scanline in the offscreen bitmap must be padded out to a multiple of 8 (inserting zeros if necessary when downloading a source bitmap to offscreen memory). Note that the BitmapStridePad member also defines the padding requirements for all buffers passed to the SetDrawBuffer function.

The `MonoBitmapStartAlign` member defines the alignment requirements in bytes for monochrome offscreen memory bitmaps for this graphics mode, which are used by the `PutMonoImageMSBLin` and `PutMonoImageLSBLin` functions. If the value in here is set to 8 for instance, then the start for all monochrome offscreen bitmaps must be aligned to an 8 byte boundary in order to be used by the `PutMonoImageMSBLin` and `PutMonoImageLSBLin` functions.

The `MonoBitmapStridePad` member defines the alignment requirements in bytes for the stride of monochrome offscreen memory bitmaps (the number of bytes to move from one line of the bitmap to the next). If the value in here is set to 8 for instance, then the number of bytes for each scanline in the monochrome offscreen bitmap must be padded out to a multiple of 8 (inserting zeros if necessary when downloading a source bitmap to offscreen memory).

The `RefreshRateList` member contains a list of all valid refresh rates supported by the display mode which can be passed to the `SetVideoMode` function. Interlaced modes are indicated by a negative refresh rate (ie: 48Hz Interlaced is -48). The current default refresh rate is stored in the `DefaultRefreshRate` member, and except for special circumstances the default refresh rate set by the user should be used rather than overriding the refresh rate.

The `BitBltCaps` member defines the extended hardware BitBlt capabilities for the graphics mode, and points to a static `GA_bltFx` structure. Refer to the documentation of `GA_bltFx` to determine what this structure contains.

The `VideoWindows` member defines the hardware video capabilities for each of up to a maximum number of hardware video overlay windows. The list of hardware video overlay window capabilities is terminated with a NULL pointer. For instance if only 2 hardware video windows are supported, the first two entries in this array would point to valid `GA_videoInf` structures, while the third would contain a NULL terminating the list. Refer to the documentation of `GA_videoInf` to determine what these structures contains.

The `HW3DCaps` member defined the hardware 3D capabilities for the graphics mode, and points to a static `GA_3DState` structure. Refer to the documentation of `GA_3DState` to determine what this structure contains.

The `LinearSize` member is the 32-bit length of the linear frame buffer memory in bytes. It can be any length up to the size of the available video memory. The `LinearBasePtr` member is the 32-bit physical address of the start of frame buffer memory when the controller is in linear frame buffer memory mode for this particular graphics mode. If the linear framebuffer is not available, then this member will be zero.

The `AttributesExt` member contains a number of extended flags that describes certain important characteristics of the graphics mode, and the values this member contains are defined in the `GA_AttributeExtFlagsType`.



The `PhysicalXResolution` and `PhysicalYResolution` specify the physical width and height in pixel elements for this display mode. The physical resolution is the resolution of all visible pixels on the display, and may be smaller than the logical resolution if the mode has hardware panning enabled.

---

**Note:** *The `LinearSize` and `LinearBasePtr` members are duplicated in the mode information block because they may possibly change locations in memory depending on the display mode. Normally applications will always use the value stored in the the `GA_devCtx` `LinearMem` pointer to directly access the framebuffer (which is automatically adjusted for you), however if the information about the framebuffer starting address needed to be reported to other applications directly, the values stored in this mode information block should be used.*

**Note:** *The memory pointed to by the `RefreshRateList`, `BitBlTCaps`, `VideoWindows` and `HW3DCaps` fields will be reused the next time `GetVideoModeInfo` is called, so do not rely on the information in these fields to remain the same across calls to this function.*

**Note:** *The `dwSize` member is intended for future compatibility, and should be set to the size of the structure as defined in this header file. Future drivers will be compatible with older software by examining this value.*

---

## Members

<i><code>dwSize</code></i>	Set to size of structure in bytes
<i><code>Attributes</code></i>	Mode attributes
<i><code>XResolution</code></i>	Logical horizontal resolution in pixels
<i><code>YResolution</code></i>	Logical vertical resolution in lines
<i><code>XCharSize</code></i>	Character cell X dimension for text modes
<i><code>YCharSize</code></i>	Character cell Y dimension for text modes
<i><code>BytesPerScanLine</code></i>	Bytes per horizontal scan line
<i><code>MaxBytesPerScanLine</code></i>	Maximum bytes per scan line
<i><code>MaxScanLineWidth</code></i>	Maximum pixels per scan line
<i><code>MaxScanLines</code></i>	Maximum number of scanlines for default scanline width
<i><code>LinearHeapStart</code></i>	Start of linear only heap (if any)
<i><code>MaxLinearOffset</code></i>	Maximum display memory offset for linear drawing
<i><code>BitsPerPixel</code></i>	Bits per pixel
<i><code>PixelFormat</code></i>	Pixel format for the display mode
<i><code>MaxBuffers</code></i>	Maximum number of display buffers
<i><code>MaxPixelClock</code></i>	Maximum pixel clock for mode
<i><code>DefaultRefreshRate</code></i>	Currently active default refresh rate
<i><code>RefreshRateList</code></i>	List of all valid refresh rates terminated with -1.
<i><code>BitmapStartAlign</code></i>	Linear bitmap start alignment in bytes
<i><code>BitmapStridePad</code></i>	Linear bitmap stride pad in bytes
<i><code>MonoBitmapStartAlign</code></i>	Linear bitmap start alignment in bytes
<i><code>MonoBitmapStridePad</code></i>	Linear bitmap stride pad in bytes
<i><code>BitBlTCaps</code></i>	Hardware 2D BitBlTfx capabilities
<i><code>VideoWindows</code></i>	Up to 8 hardware video overlays

<i>HW3DCaps</i>	Hardware 3D capabilities
<i>MaxOOZ</i>	Maximum ooz coordinate value in floating point
<i>MaxOOW</i>	Maximum oow coordinate value in floating point
<i>MaxOOS</i>	Maximum one over s coordinate value in floating point
<i>MaxOOT</i>	Maximum one over t coordinate value in floating point
<i>DepthFormats</i>	Depth buffer formats flags
<i>DepthStartAlign</i>	Depth buffer start alignment in bytes
<i>DepthStridePad</i>	Depth buffer stride pad in bytes
<i>TextureFormats</i>	Texture formats flags
<i>TextureStartAlign</i>	Texture start alignment in bytes
<i>TextureStridePad</i>	Texture stride pad in bytes
<i>TextureMaxX</i>	Maximum texture X dimension
<i>TextureMaxY</i>	Maximum texture Y dimension
<i>TextureMaxAspect</i>	Maximum texture aspect ratio (1:x)
<i>StencilFormats</i>	Stencil buffer formats flags
<i>StencilStartAlign</i>	Stencil buffer start alignment in bytes
<i>StencilStridePad</i>	Stencil buffer stride pad in bytes
<i>LinearSize</i>	Linear buffer size in bytes
<i>LinearBasePtr</i>	Physical addr of linear buffer
<i>AttributesExt</i>	Extended mode attributes flags
<i>PhysicalXResolution</i>	Physical horizontal resolution in pixels
<i>PhysicalYResolution</i>	Physical vertical resolution in lines

## GA\_modeProfile

### Declaration

```
typedef struct {
    N_uint32    dwSize;
    struct {
        N_uint8 numModes;
        GA_mode modeList[256];
    } m;
    struct {
        N_uint8 numModes;
        GA_mode modeList[256];
    } vbe;
} GA_modeProfile
```

### Prototype In

snap/graphics.h

### Description

Structure returned by GetModeProfile, which contains configuration information about the mode profile for the installed device driver. A default mode profile is shipped with the graphics device drivers, but a new mode profile can be downloaded at any time (to implement new display modes using the new Dial-A-Mode interface). Note that a mode must also have the associated CRTC tables before it will work.

---

**Note:** *The dwSize member is intended for future compatibility, and should be set to the size of the structure as defined in this header file. Future drivers will be compatible with older software by examining this value.*

---

### Members

<i>dwSize</i>	Set to size of structure in bytes
<i>numModes</i>	Count for the number of configured display modes
<i>modeList</i>	Array of up to 256 configured display modes
<i>numModes</i>	Count for the number of configured display modes
<i>vbeModeList</i>	Array of up to 256 modes reported to the VBE driver

## GA\_monitor

### Declaration

```
typedef struct {
    char    mfr[MONITOR_MFR_LEN+1];
    char    model[MONITOR_MODEL_LEN+1];
    char    PNPID[8];
    uchar   maxResolution;
    uchar   minHScan;
    uchar   maxHScan;
    uchar   minVScan;
    uchar   maxVScan;
    uchar   flags;
} GA_monitor
```

### Prototype In

snap/monitor.h

### Description

Monitor configuration information structure. This structure defines the capabilities of the attached display monitor, and is used internally in SNAP Graphics to decide what features the driver should report for the attached monitor.

### Members

<i>mfr</i>	Monitor manufacturer (key)
<i>model</i>	Monitor model name (sub-key)
<i>PNPID</i>	Plug and Play ID (optional)
<i>maxResolution</i>	Maximum resolution id
<i>minHScan</i>	Minimum horizontal scan
<i>maxHScan</i>	Maximum horizontal scan
<i>minVScan</i>	Minimum vertical scan
<i>maxVScan</i>	Maximum vertical scan
<i>flags</i>	Capabilities flags

## GA\_monitorFlagsType

### Declaration

```
typedef enum {
    Monitor_DPMSEnabled    = 0x01,
    Monitor_GTFEnabled     = 0x02,
    Monitor_FixedFreq      = 0x04,
    Monitor_HSyncNeg       = 0x08,
    Monitor_VSyncNeg       = 0x10,
    Monitor_16to9          = 0x20,
    Monitor_Exclude4to3    = 0x40,
    Monitor_Changed        = 0x80
} GA_monitorFlagsType
```

### Prototype In

snap/monitor.h

### Description

This enumeration defines the flags for the capabilities for monitors as defined in the *GA\_monitor* record.

### Members

<i>Monitor_DPMSEnabled</i>	Monitor supports DPMS Power Management
<i>Monitor_GTFEnabled</i>	Monitor supports VESA Generalised Timing Formula
<i>Monitor_FixedFreq</i>	Monitor is a fixed frequency monitor
<i>Monitor_HSyncNeg</i>	HSync- is required for fixed frequency
<i>Monitor_VSyncNeg</i>	VSync- is required for fixed frequency
<i>Monitor_16to9</i>	Monitor supports 16:9 aspect ratio modes
<i>Monitor_Exclude4to3</i>	Driver should exclude all 4:3 aspect modes
<i>Monitor_Changed</i>	Monitor record has been changed

## GA\_monoCursor

### Declaration

```
typedef struct {
    N_uint8    XORMask[512];
    N_uint8    ANDMask[512];
    N_uint32    HotX;
    N_uint32    HotY;
} GA_monoCursor
```

### Prototype In

snap/graphics.h

### Description

Hardware monochrome cursor structure. This structure defines a monochrome hardware cursor that is downloaded to the hardware. The cursor is defined as a 64x64 image with an AND and XOR mask. The definition of the AND mask, XOR mask and the pixels that will appear on the screen is as follows (same as the Microsoft Windows cursor format):

```
AND XOR Result
0  0  Transparent (color from screen memory)
0  1  Invert (complement of color from screen memory)
1  0  Cursor background color
1  1  Cursor foreground color
```

The HotX and HotY members define the *hot spot* for the cursor, which is the location where the logical mouse pointer is located in the cursor image. When you click the mouse, the pixel under the hot-spot is the pixel selected.

### Members

<i>XORMask</i>	Cursor XOR mask
<i>ANDMask</i>	Cursor AND mask
<i>HotX</i>	Cursor X coordinate hot spot
<i>HotY</i>	Cursor Y coordinate hot spot

## GA\_multiHeadType

### Declaration

```
typedef enum {
    gaActiveHeadClone      = -1,
    gaActiveHeadPrimary    = 0,
    gaActiveHeadSecondary  = 1,
    gaActiveHeadTernary    = 2
} GA_multiHeadType
```

### Prototype In

snap/graphics.h

### Description

This enumeration defines the flags stored in the bMultiDisplay field of the *GA\_options* structure.

### Members

<i>gaActiveHeadClone</i>	Active head is cloned across all displays (default)
<i>gaActiveHeadPrimary</i>	Active head is primary display
<i>gaActiveHeadSecondary</i>	Active head is secondary display
<i>gaActiveHeadTernary</i>	Active head is third display

## GA\_options

### Declaration

```
typedef struct {
    N_uint32      dwSize;
    N_fix32      memoryClock;
    N_fix32      defaultMemoryClock;
    N_fix32      maxMemoryClock;
    GA_paletteExt gammaRamp[256];
    N_int32      outputDevice;
    GA_TVParams  TV640PALUnder;
    GA_TVParams  TV640NTSCUnder;
    GA_TVParams  TV640PALOver;
    GA_TVParams  TV640NTSCOver;
    GA_TVParams  TV800PALUnder;
    GA_TVParams  TV800NTSCUnder;
    GA_TVParams  TV800PALOver;
    GA_TVParams  TV800NTSCOver;
    N_uint8      bRes1;
    N_uint8      bRes2;
    N_int32      RTCFrequency;
    N_int32      RTCAdvanceTicks;
    N_uint8      bRes3;
    N_uint16     ioPort;
    N_uint8      ioAndMask;
    N_uint8      ioLeftOrMask;
    N_uint8      ioRightOrMask;
    N_uint8      ioOffOrMask;
    N_uint8      vSyncWidthLeft;
    N_uint8      vSyncWidthRight;
    N_uint8      text80x43Height;
    N_uint8      text80x50Height;
    N_uint8      text80x60Height;
    N_uint8      text100x43Height;
    N_uint8      text100x50Height;
    N_uint8      text100x60Height;
    N_uint8      text132x43Height;
    N_uint8      text132x50Height;
    N_uint8      text132x60Height;
    N_uint8      tripleScanLowRes;
    N_uint8      doubleScan512;
    N_uint8      stereoRefresh;
    N_uint8      stereoRefreshInterlaced;
    N_uint8      stereoMode;
    N_uint8      stereoModeWindowed;
    N_uint8      stereoBlankInterval;
    N_uint8      stereoRefreshWindowed;
    N_uint8      stereoRefreshWindowedInterlaced;
    N_uint32     stereoDevice;
    N_uint16     glassesType;
    N_uint16     stereoBlankIntervalPercent;
    N_fix32      engineClock;
    N_fix32      defaultEngineClock;
    N_fix32      maxEngineClock;
    N_uint8      stereoControlPanelOptions;
    N_uint8      stereoCursorRedraw;
    N_uint8      res0[14];
    N_int16      LCDPanelWidth;
    N_int16      LCDPanelHeight;
    N_uint8      bLCDExpand;
    N_uint8      bPrefer16bpp;
}
```



N_uint8	bPrefer32bpp;
N_int16	TVMaxWidth;
N_int16	TVMaxHeight;
N_uint8	res1[95];
N_uint32	resolutions[GA_MAX_RESOLUTIONS];
N_uint8	colorDepths[GA_MAX_COLORDEPTHS];
N_uint8	maxRefresh;
N_uint8	accelType;
N_uint8	res2[159];
N_uint8	bDebugMode;
N_uint8	bGenericRefresh;
N_uint8	bDialAMode;
N_uint8	bVirtualScroll;
N_uint8	bDoubleBuffer;
N_uint8	bTripleBuffer;
N_uint8	bHardwareStereoSync;
N_uint8	bStereo;
N_uint8	bMultiDisplay;
N_uint8	bPortrait;
N_uint8	bFlipped;
N_uint8	bInvertColors;
N_uint8	bReserved1;
N_uint8	bReserved2;
N_uint8	bVirtualDisplay;
N_uint8	bAGPFastWrite;
N_uint8	bZoom;
N_uint8	bMultiHead;
N_uint8	res3[154];
N_uint8	bTVOut;
N_uint8	bTVTuner;
N_uint8	bDualHead;
N_uint8	bDPMS;
N_uint8	bDDC;
N_uint8	bDDCCI;
N_uint8	bGammaCorrect;
N_uint8	bHardwareCursor;
N_uint8	bHardwareColorCursor;
N_uint8	bHardwareVideo;
N_uint8	bHardwareAccel2D;
N_uint8	bHardwareAccel3D;
N_uint8	bMonoPattern;
N_uint8	bTransMonoPattern;
N_uint8	bColorPattern;
N_uint8	bTransColorPattern;
N_uint8	bSysMem;
N_uint8	bLinear;
N_uint8	bBusMaster;
N_uint8	bDrawScanList;
N_uint8	bDrawEllipseList;
N_uint8	bDrawFatEllipseList;
N_uint8	bDrawRect;
N_uint8	bDrawRectLin;
N_uint8	bDrawTrap;
N_uint8	bDrawLine;
N_uint8	bDrawStippleLine;
N_uint8	bPutMonoImage;
N_uint8	bClipMonoImage;
N_uint8	bBitBlt;
N_uint8	bBitBltPatt;
N_uint8	bBitBltColorPatt;
N_uint8	bSrcTransBlt;
N_uint8	bDstTransBlt;
N_uint8	bStretchBlt;

```

N_uint8      bConvertBlt;
N_uint8      bStretchConvertBlt;
N_uint8      bBitBltFx;
N_uint8      bGetBitmap;
N_uint8      res4[256];
GA_layout    multiHeadSize;
GA_layout    multiHeadRes[GA_MAX_HEADS];
GA_layout    multiHeadBounds[GA_MAX_HEADS];
} GA_options

```

## Prototype In

snap/graphics.h

## Description

Structure returned by `GetOptions`, which contains configuration information about the options for the installed device driver. All the boolean configuration options are enabled by default and can be optionally turned off by the user via the configuration functions.

This structure also contains the configuration information for the software stereo page flipping support in SNAP Graphics.

If you select the `gaGlassesIOPort` type, then you need to fill in the `ioPort`, `ioAndMask`, `ioLeftOrMask`, `ioRightOrMask` and `ioOffOrMask` fields. These fields define the values used to toggle the specified I/O port when the glasses need to be flipped. First the existing value is read from the specified I/O port, the AND mask is applied and then the appropriate OR mask is applied depending on the state of the glasses. This value is then written back to the desired I/O port.

If you select the `gaGlassesGenericVSync` type, then you need to fill in the `VSyncWidthLeft` and `VSyncWidthRight` fields, which define the vertical sync width to program when the desired eye should be active.

The values in the `RTCFrequency` and `RTCAdvanceTicks` define the frequency of the stereo timer interrupt, which can be used to fine tune the overheads taken by the stereo page flip handler for maximum performance before stuttering begins (ie: lost frames). The `RTCFrequency` field can be any power of 2 frequency between 1024Hz and 8192Hz, and the `RTCAdvanceTicks` should be a value larger than 1. For most systems an `RTCFrequency` value of 2048 and an `RTCAdvanceTicks` of 2 will work well. SNAP Graphics will however choose good defaults for the target OS if these values are not overridden.

---

**Note:** *The `dwSize` member is intended for future compatibility, and should be set to the size of the structure as defined in this header file. Future drivers will be compatible with older software by examining this value.*

---

## Members

<i>dwSize</i>	Set to size of structure in bytes
<i>memoryClock</i>	Currently configured memory clock
<i>defaultMemoryClock</i>	Current hardware default memory clock
<i>maxMemoryClock</i>	Maximum allowable memory clock

<i>gammaRamp</i>	Default gamma ramp for RGB display modes
<i>outputDevice</i>	Currently configured output device
<i>TV640PALUnder</i>	TV parameters for 640x480 PAL underscan modes
<i>TV640NTSCUnder</i>	TV parameters for 640x480 NTSC underscan modes
<i>TV640PALOver</i>	TV parameters for 640x480 PAL overscan modes
<i>TV640NTSCOver</i>	TV parameters for 640x480 NTSC overscan modes
<i>TV800PALUnder</i>	TV parameters for 800x600 PAL underscan modes
<i>TV800NTSCUnder</i>	TV parameters for 800x600 NTSC underscan modes
<i>TV800PALOver</i>	TV parameters for 800x600 PAL overscan modes
<i>TV800NTSCOver</i>	TV parameters for 800x600 NTSC overscan modes
<i>numHorzDisplay</i>	Number of horizontal displays
<i>numVertDisplay</i>	Number of vertical displays
<i>RTCFrequency</i>	Frequency for real time clock for software stereo
<i>RTCAvanceTicks</i>	Number of ticks to advance for software stereo
<i>glassesType</i>	Type of stereo glasses defined by GA_glassesTypeFlags
<i>ioPort</i>	Generic I/O port for controlling glasses
<i>ioAndMask</i>	I/O port AND mask
<i>ioLeftOrMask</i>	I/O port OR mask when left eye is active
<i>ioRightOrMask</i>	I/O port OR mask when right eye is active
<i>ioOffOrMask</i>	I/O port OR mask when glasses are off
<i>vSyncWidthLeft</i>	Vertical sync width when left eye is active
<i>vSyncWidthRight</i>	Vertical sync width when right eye is active
<i>text80x43Height</i>	Character height for 80x43 text mode (8,14 or 16)
<i>text80x50Height</i>	Character height for 80x50 text mode (8,14 or 16)
<i>text80x60Height</i>	Character height for 80x60 text mode (8,14 or 16)
<i>text100x43Height</i>	Character height for 100x43 text mode (8,14 or 16)
<i>text100x50Height</i>	Character height for 100x50 text mode (8,14 or 16)
<i>text100x60Height</i>	Character height for 100x60 text mode (8,14 or 16)
<i>text132x43Height</i>	Character height for 132x43 text mode (8,14 or 16)
<i>text132x50Height</i>	Character height for 132x50 text mode (8,14 or 16)

<i>text132x60Height</i>	16) Character height for 132x60 text mode (8,14 or 16)
<i>tripleScanLowRes</i>	True to triple scan low res modes
<i>doubleScan512</i>	True to double scan 512x384 modes
<i>stereoRefresh</i>	Value to use for stereo mode refresh rate
<i>stereoRefreshInterlaced</i>	Value to use for stereo mode interlaced refresh rate
<i>stereoMode</i>	Stereo mode to be used for fullscreen applications
<i>stereoModeWindowed</i>	Stereo mode to be used for windowed applications
<i>stereoBlankInterval</i>	Stereo mode blank interval for above below format
<i>stereoDevice</i>	Stereo device ID defined by stereo control panel
<i>engineClock</i>	Currently configured graphics engine clock
<i>defaultEngineClock</i>	Current hardware default graphics engine clock
<i>maxEngineClock</i>	Maximum allowable graphics engine clock
<i>LCDPanelWidth</i>	Width of attached LCD panel in pixels
<i>LCDPanelHeight</i>	Height of attached LCD panel in lines
<i>bLCDExpand</i>	Enable expansion of modes to fill LCD panel
<i>bDebugMode</i>	Enable debug log filter driver (0 is off)
<i>bGenericRefresh</i>	Enable generic refresh rate control
<i>bDialAMode</i>	Enable Dial-A-Mode generic mode interface
<i>bVirtualScroll</i>	Enable virtual scrolling functions
<i>bDoubleBuffer</i>	Enable double buffering functions
<i>bTripleBuffer</i>	Enable triple buffering functions
<i>bHardwareStereoSync</i>	Enable hardware stereo sync flag
<i>bStereo</i>	Enable stereo display mode support
<i>bMultiDisplay</i>	Enable multiple display mode support
<i>bPortrait</i>	Enable portrait display mode
<i>bFlipped</i>	Enable flipped display mode
<i>bInvertColors</i>	Enable invert color mode
<i>bVirtualDisplay</i>	Enable virtual display mode
<i>bAGPFastWrite</i>	Enable AGP fast write (only here to be licensed)
<i>bZoom</i>	Enable zoom support
<i>bMultiHead</i>	Enable multi-head support
<i>bTVOut</i>	Enable TV Output support
<i>bTVTuner</i>	Enable TV Tuner support
<i>bDualHead</i>	Enable Dual Head CRTC support
<i>bDPMS</i>	Enable DPMS Display Power Management support
<i>bDDC</i>	Enable DDC Display Data Channel functions
<i>bDDCCI</i>	Enable DDC/CI Control Interface functions

<i>bGammaCorrect</i>	Enable gamma correction
<i>bHardwareCursor</i>	Enable hardware cursor
<i>bHardwareVideo</i>	Enable hardware video
<i>bHardwareAccel2D</i>	Enable hardware 2D acceleration
<i>bHardwareAccel3D</i>	Enable hardware 2D acceleration
<i>bMonoPattern</i>	Enable 8x8 mono pattern fills
<i>bTransMonoPattern</i>	Enable 8x8 mono transparent pattern fills
<i>bColorPattern</i>	Enable 8x8 color pattern fills
<i>bTransColorPattern</i>	Enable 8x8 color transparent pattern fills
<i>bSysMem</i>	Enable system memory blits
<i>bLinear</i>	Enable linear offscreen memory blits
<i>bBusMaster</i>	Enable bus mastering functions
<i>bDrawScanList</i>	Enable DrawScanList family of functions
<i>bDrawEllipseList</i>	Enable DrawEllipseList family of functions
<i>bDrawFatEllipseList</i>	Enable DrawFatEllipseList family of functions
<i>bDrawRect</i>	Enable DrawRect family of functions
<i>bDrawRectLin</i>	Enable DrawRectLin family of functions
<i>bDrawTrap</i>	Enable DrawTrap family of functions
<i>bDrawLine</i>	Enable DrawLine function
<i>bDrawStippleLine</i>	Enable DrawStippleLine function
<i>bPutMonoImage</i>	Enable PutMonoImage family of functions
<i>bClipMonoImage</i>	Enable ClipMonoImage family of functions
<i>bBitBlt</i>	Enable BitBlt family of functions
<i>bBitBltPatt</i>	Enable BitBltPatt family of functions
<i>bBitBltColorPatt</i>	Enable BitBltColorPatt family of functions
<i>bSrcTransBlt</i>	Enable SrcTransBlt family of functions
<i>bDstTransBlt</i>	Enable DstTransBlt family of functions
<i>bStretchBlt</i>	Enable StretchBlt family of functions
<i>bConvertBlt</i>	Enable ConvertBlt family of functions
<i>bStretchConvertBlt</i>	Enable StretchConvertBlt family of functions
<i>bBitBltFx</i>	Enable BitBltFx family of functions
<i>bGetBitmap</i>	Enable GetBitmap family of functions
<i>multiHeadSize</i>	Virtual size for multi-head displays
<i>multiHeadRes</i>	Physical resolutions for multi-head displays
<i>multiHeadBounds</i>	Virtual layout for multi-head displays

## GA\_palette

### Declaration

```
typedef struct {  
    N_uint8    Blue;  
    N_uint8    Green;  
    N_uint8    Red;  
    N_uint8    Alpha;  
} GA_palette
```

### Prototype In

snap/graphics.h

### Description

Palette entry structure, which defines a single entry in the hardware color lookup table or gamma correction table.

### Members

<i>blue</i>	Blue component of palette entry, range [0-255]
<i>green</i>	Green component of palette entry, range [0-255]
<i>red</i>	Blue component of palette entry, range [0-255]
<i>alpha</i>	Alpha or alignment byte

## GA\_paletteExt

### Declaration

```
typedef struct {  
    N_uint16    Blue;  
    N_uint16    Green;  
    N_uint16    Red;  
} GA_paletteExt
```

### Prototype In

snap/graphics.h

### Description

Extended palette entry structure, which defines a single entry in the hardware color lookup table or gamma correction table.

### Members

<i>blue</i>	Blue component of palette entry, range [0-65535]
<i>green</i>	Green component of palette entry, range [0-65535]
<i>red</i>	Blue component of palette entry, range [0-65535]

## GA\_pattern

### Declaration

```
typedef struct {  
    N_uint8    p[8];  
} GA_pattern
```

### Prototype In

snap/graphics.h

### Description

Fundamental type definition for an array element of an 8x8 monochrome pattern. Each line in the pattern is represented as a single byte, with 8 bytes in total for the entire pattern.

### Members

*p*            8 bytes of pattern data



## GA\_pixelFormat

### Declaration

```
typedef struct {
    N_uint8 RedMask;
    N_uint8 RedPosition;
    N_uint8 RedAdjust;
    N_uint8 GreenMask;
    N_uint8 GreenPosition;
    N_uint8 GreenAdjust;
    N_uint8 BlueMask;
    N_uint8 BluePosition;
    N_uint8 BlueAdjust;
    N_uint8 AlphaMask;
    N_uint8 AlphaPosition;
    N_uint8 AlphaAdjust;
} GA_pixelFormat
```

### Prototype In

snap/graphics.h

### Description

Structure representing the format of an RGB pixel. This structure is used to describe the RGB pixel format for SNAP graphics modes, as well as the pixel format for system memory buffers converted on the fly by SNAP Graphics to the destination pixel format. RGB pixel formats are required for pixel depths greater than or equal to 15-bits per pixel. The pixel formats for 15 and 16-bit modes are constant and never change, however there are 2 possible pixel formats for 24 bit RGB modes and 4 possible formats for 32 bit RGB modes that are supported by the MGL. The possible modes for 24-bits per pixel are:

<i>24-bit</i>	Description
<i>RGB</i>	Values are packed with Red in byte 2, Green in byte 1 and Blue in byte 0. This is the standard format used by all 24 bit Windows BMP files, and the native display format for most graphics hardware on the PC.
<i>BGR</i>	Values are packed with Blue in byte 2, Green in byte 1 and Red in byte 0. This format is the native display format for some graphics hardware on the PC.

The possible modes for 32-bits per pixel are:

<i>32-bit</i>	Description
<i>ARGB</i>	Values are packed with Red in byte 2, Green in byte 1 and Blue in byte 0 and alpha in byte 3.
<i>ABGR</i>	Values are packed with Blue in byte 2, Green in byte 1 and Red in byte 0 and alpha in byte 3.
<i>RGBA</i>	Values are packed with Red in byte 3, Green in byte 2 and Blue in byte 1 and alpha in byte 0.
<i>BGRA</i>	Values are packed with Blue in byte 3, Green in byte 2 and Red in byte 1 and alpha in byte 0.

If you intend to write your own direct rendering code for RGB graphics modes, you will need to write your code so that it will adapt to the underlying pixel format used by the

hardware to display the correct colors on the screen. SNAP Graphics has the ability to perform pixel format translation on the fly using the ConvertBlt family of functions, but this can be time consuming so directly rendering in the native pixel format can be more efficient. The formula for packing the pixel data into the proper positions given three 8-bit RGB values is as follows:

```
color = ((GA_color)((R >> RedAdjust) & RedMask)
        << RedPosition)
        | ((GA_color)((G >> GreenAdjust) & GreenMask)
        << GreenPosition)
        | ((GA_color)((B >> BlueAdjust) & BlueMask)
        << BluePosition);
```

Alternatively you can unpack the color values from the framebuffer with the following code (note that you lose precision when unpacking values from the framebuffer since the bottom bits always get set to 0):

```
R = (((color) >> RedPosition) & RedMask)
    << RedAdjust;
G = (((color) >> GreenPosition) & GreenMask)
    << GreenAdjust;
B = (((color) >> BluePosition) & BlueMask)
    << BlueAdjust;
```

If you wish to create your own pixel formats (such as to create memory custom bitmaps), the following list defines all the pixel formats that the SNAP Graphics knows how to deal with:

```
{0x1F,0x0A,3, 0x1F,0x05,3, 0x1F,0x00,3, 0x01,0x0F,7}, // 555 15bpp
{0x1F,0x0B,3, 0x3F,0x05,2, 0x1F,0x00,3, 0x00,0x00,0}, // 565 16bpp
{0xFF,0x10,0, 0xFF,0x08,0, 0xFF,0x00,0, 0x00,0x00,0}, // RGB 24bpp
{0xFF,0x00,0, 0xFF,0x08,0, 0xFF,0x10,0, 0x00,0x00,0}, // BGR 24bpp
{0xFF,0x10,0, 0xFF,0x08,0, 0xFF,0x00,0, 0xFF,0x18,0}, // ARGB 32bpp
{0xFF,0x00,0, 0xFF,0x08,0, 0xFF,0x10,0, 0xFF,0x18,0}, // ABGR 32bpp
{0xFF,0x18,0, 0xFF,0x10,0, 0xFF,0x08,0, 0xFF,0x00,0}, // RGBA 32bpp
{0xFF,0x08,0, 0xFF,0x10,0, 0xFF,0x18,0, 0xFF,0x00,0}, // BGRA 32bpp
```

One special cased pixel format is used to represent and 8bpp color index bitmap with an alpha channel. This pixel format is a 16-bit wide pixel format, but the red channel is considered to contain the 8-bit color index values. The pixel format structure for this type of bitmap looks like the following:

```
{0xFF,0x00,0, 0x00,0x00,0, 0x00,0x00,0, 0xFF,0x08,0}, // A8CI 8bpp + Alpha
```

## Members

<b>RedMask</b>	Unshifted 8-bit mask for the red color channel
<b>RedPosition</b>	Bit position for bit 0 of the red color channel information
<b>RedAdjust</b>	Number of bits to shift the 8-bit red value right
<b>GreenMask</b>	Unshifted 8-bit mask for the green color channel
<b>GreenPosition</b>	Bit position for bit 0 of the green color channel information
<b>GreenAdjust</b>	Number of bits to shift the 8-bit green value right
<b>BlueMask</b>	Unshifted 8-bit mask for the blue color channel

<i>BluePosition</i>	Bit position for bit 0 of the blue color channel information
<i>BlueAdjust</i>	Number of bits to shift the 8-bit blue value right
<i>AlphaMask</i>	Unshifted 8-bit mask for the alpha channel
<i>AlphaPosition</i>	Bit position for bit 0 of the alpha channel information
<i>AlphaAdjust</i>	Number of bits to shift the 8-bit alpha value right

## GA\_recMode

### Declaration

```
typedef struct {
    N_uint16    XResolution;
    N_uint16    YResolution;
    N_uint8     BitsPerPixel;
    N_int8      RefreshRate;
} GA_recMode
```

### Prototype In

snap/graphics.h

### Description

Structure used to store the recommended modes for SNAP Graphics.

### Members

<i>XResolution</i>	X resolution for the recommended mode
<i>YResolution</i>	Y resolution for the recommended mode
<i>BitsPerPixel</i>	BitsPerPixel for the recommended mode
<i>RefreshRate</i>	RefreshRate for the recommended mode (up to 127Hz)

## GA\_rect

### Declaration

```
typedef struct {  
    N_int32 left;  
    N_int32 top;  
    N_int32 right;  
    N_int32 bottom;  
} GA_rect
```

### Prototype In

snap/graphics.h

### Description

Fundamental type definition for an integer rectangle. Note that SNAP Graphics defines and uses rectangles such that the bottom and right coordinates are not actually included in the pixels that define a raster coordinate rectangle. This allows for correct handling of overlapping rectangles without drawing any pixels twice.

### Members

<i>left</i>	Left coordinate of the rectangle
<i>top</i>	Top coordinate of the rectangle
<i>right</i>	Right coordinate of the rectangle
<i>bottom</i>	Bottom coordinate of the rectangle

## GA\_region

### Declaration

```
typedef struct {  
    GA_rect rect;  
    GA_span *spans;  
} GA_region
```

### Prototype In

snap/graphics.h

### Description

Fundamental type definition representing a complex region. Complex regions are used to represent non-rectangular areas as unions of smaller rectangles (the smallest being a single pixel). You can use complex regions to build complex clipping regions for user interface library development.

If the Spans field for the region is NULL, then the region is a simple region and is composed of only a single rectangle. Note however that you can have a simple region that consists of only single rectangle in the span structure (usually after complex region arithmetic). You can use the GA\_IsSimpleRegion macro to determine if the region contains only a single rectangle.

### Members

<i>Rect</i>	Bounding rectangle for the region
<i>spans</i>	Pointer to the internal region span structure.

## GA\_regionFuncs

**Prototype In**

snap/graphics.h

**Description**

Function group containing all region management functions available via the SNAP API's. These functions may be used externally for complex region management, but they are also used extensively by SNAP Graphics for offscreen buffer management.

Note also that this function group is *only* returned by the 2D reference rasteriser library, and not by hardware drivers.

---

**Note:** *Be sure to fill in the dwSize member of this structure when you call GA\_queryFunctions to the correct size of the structure at compile time!*

---

## ClearRegion

Clears the specified region to an empty region.

### Declaration

```
void NAPI GA_regionFuncs::ClearRegion(  
    GA_region *r)
```

### Prototype In

snap/graphics.h

### Parameters

*r* region to be cleared

### Description

This function clears the specified region to an empty region, freeing up all the memory used to store the region data.

### See Also

*NewRegion, CopyRegion, FreeRegion*



## CopyIntoRegion

Copy the contents of one region into another region.

### Declaration

```
void NAPI GA_regionFuncs::CopyIntoRegion(  
    GA_region *d,  
    const GA_region *s)
```

### Prototype In

snap/graphics.h

### Parameters

<i>d</i>	Pointer to destination region
<i>s</i>	Pointer to source region

### Description

Copies the definition for an entire region into the destination region, clearing any region information already present in the destination. This function is similar to *CopyRegion*, however it does not allocate a new region but rather copies the data into an existing region.

### See Also

*NewRegion*, *FreeRegion*, *ClearRegion*, *CopyRegion*

## CopyRegion

Create a copy of the specified region.

### Declaration

```
GA_region * NAPI GA_regionFuncs::CopyRegion(  
    const GA_region *s)
```

### Prototype In

snap/graphics.h

### Parameters

*s*                      Pointer to source region

### Return Value

Pointer to the copied region, or NULL if out of memory.

### Description

Copies the definition for an entire region and returns a pointer to the newly created region. The space for the copied region is allocated from the region memory pool, which SNAP uses to maintain a local memory allocation scheme for regions to increase performance.

If there is not enough memory to copy the region, this routine will return NULL.

### See Also

*NewRegion, FreeRegion, ClearRegion, CopyIntoRegion*

## DiffRegion

Compute the Boolean difference of two regions.

### Declaration

```
ibool NAPI GA_regionFuncs::DiffRegion(  
    GA_region *r1,  
    const GA_region *r2)
```

### Prototype In

snap/graphics.h

### Parameters

- |           |   |
|-----------|---|
| <i>r1</i> | Region from which r2 is subtracted, which also becomes the result region. |
| <i>r2</i> | Region to be subtracted from r1   |

### Return Value

True if the difference is valid, false if an empty region was created.

### Description

Computes the Boolean difference of two regions by subtracting the area covered by region r2 from region r1, computing the resulting region in r1, which may result in an empty region. If you need to retain the value of r1, you need to first copy r1 to a temporary region.

### See Also

*DiffRegionRect, UnionRegion, SectRegion*

## DiffRegionRect

Compute the Boolean difference of a region and a rectangle.

### Declaration

```
ibool NAPI GA_regionFuncs::DiffRegionRect(
    GA_region *r1,
    const GA_rect *r2)
```

### Prototype In

snap/graphics.h

### Parameters

<i>r1</i>	Region from which r2 is subtracted, which also becomes the result region.
<i>r2</i>	Rectangle to be subtracted from r1

### Return Value

True if the difference is valid, false if an empty region was created.

### Description

Computes the Boolean difference of a region and a simple rectangle by subtracting the area covered by rectangle r2 from region r1, computing the resulting region in r1, which may result in an empty region. If you need to retain the value of r1, you need to first copy r1 to a temporary region.

This routine will produce a simple region with only a single bounding rectangle if the original region was also a simple rectangle and the resulting region is also a single rectangle, which makes it more efficient if the region to be subtracted is a rectangle.

### See Also

*DiffRegion, UnionRegion, SectRegion*

## FreeRegion

Frees all the memory allocated by the complex region.

### Declaration

```
void NAPI GA_regionFuncs::FreeRegion(  
    GA_region *r)
```

### Prototype In

snap/graphics.h

### Parameters

*r*                      Pointer to the region to free

### Description

Frees all the memory allocated by the complex region. When you are finished with a complex region you must free it to free up the memory used to represent the union of rectangles.

### See Also

*NewRegion*, *CopyRegion*

## IsEmptyRegion

Determines if a region is empty.

### Declaration

```
ibool NAPI GA_regionFuncs::IsEmptyRegion(  
    const GA_region *r)
```

### Prototype In

snap/graphics.h

### Parameters

*r*            region to test

### Return Value

True if region is empty, false if not.

### Description

Determines if a region is empty or not. A region is defined as being empty if the bounding rectangle's right coordinate is less than or equal to the left coordinate, or if the bottom coordinate is less than or equal to the top coordinate.

### See Also

*IsEqualRegion, UnionRegion, DiffRegion, SectRegion, OffsetRegion, PtInRegion*

## IsEqualRegion

Determines if two regions are equal.

### Declaration

```
ibool NAPI GA_regionFuncs::IsEqualRegion(  
    const GA_region *r1,  
    const GA_region *r2)
```

### Prototype In

snap/graphics.h

### Parameters

<i>r1</i>	First region to compare
<i>r2</i>	Second region to compare

### Return Value

True if the regions are equal, false if not.

### Description

Determines if two regions are equal, by comparing the bounding rectangles and the definitions for both of the regions.

### See Also

*IsEmptyRegion, UnionRegion, DiffRegion, SectRegion, OffsetRegion, PtInRegion*

## NewRectRegion

Allocate a new complex region and initialises it to a specific rectangle.

### Declaration

```
GA_region * NAPI GA_regionFuncs::NewRectRegion(
    N_int32 left,
    N_int32 top,
    N_int32 right,
    N_int32 bottom)
```

### Prototype In

snap/graphics.h

### Parameters

<i>left</i>	Left coordinate of the rectangle
<i>top</i>	Top coordinate of the rectangle
<i>right</i>	Right coordinate of the rectangle
<i>bottom</i>	Bottom coordinate of the rectangle

### Return Value

New region generated, NULL if out of memory.

### Description

Allocates a new complex region, and initialises it to the passed in rectangle coordinates when first created.

### See Also

*NewRegion, FreeRegion, UnionRegion, DiffRegion, SectRegion*



## NewRegion

Allocate a new complex region.

### Declaration

```
GA_region * NAPI GA_regionFuncs::NewRegion(void)
```

### Prototype In

snap/graphics.h

### Return Value

Pointer to the new region, NULL if out of memory.

### Description

Allocates a new complex region. The new region is empty when first created. Note that SNAP maintains a local memory pool for all region allocations in order to provide the maximum speed and minimum memory overheads for region allocations.

### See Also

*NewRectRegion, FreeRegion, UnionRegion, DiffRegion, SectRegion*

## OffsetRegion

Offsets a region by the specified amount.

### Declaration

```
void NAPI GA_regionFuncs::OffsetRegion(  
    GA_region *r,  
    N_int32 dx,  
    N_int32 dy)
```

### Prototype In

snap/graphics.h

### Parameters

<i>r</i>	Region to offset
<i>dx</i>	Amount to offset x coordinates by
<i>dy</i>	Amount to offset y coordinates by

### Description

This function offsets the specified region by the dx and dy coordinates, by modifying all the coordinate locations for every rectangle in the union of rectangles that constitutes the region by the specified coordinates.

### See Also

*UnionRegion, DiffRegion, SectRegion*

## OptimizeRegion

Optimizes the union of rectangles in the region to the minimum number of rectangles.

### Declaration

```
void NAPI GA_regionFuncs::OptimizeRegion(  
    GA_region *r)
```

### Prototype In

snap/graphics.h

### Parameters

*r*                      Region to optimize

### Description

This function optimizes the specified region by traversing the region structure looking for identical spans in the region. The region algebra functions (*UnionRegion*, *DiffRegion*, *SectRegion* etc.) do not fully optimize the resulting region to save time, so after you have created a complex region you may wish to call this routine to optimize it.

Optimizing the region will find the minimum number of rectangles required to represent that region, and will result in faster drawing and traversing of the resulting region.

### See Also

*UnionRegion*, *DiffRegion*, *SectRegion*

## PtInRegion

Determines if a point is contained in a specified region.

### Declaration

```
ibool NAPI GA_regionFuncs::PtInRegion(  
    const GA_region *r,  
    N_int32 x,  
    N_int32 y)
```

### Prototype In

snap/graphics.h

### Parameters

<i>r</i>	Region to test
<i>x</i>	x coordinate to test for inclusion
<i>y</i>	y coordinate to test for inclusion

### Return Value

True if the point is contained in the region, false if not.

### Description

This function determines if a specified point is contained within a particular region.

Note that if the region has a hole it in, and the point lies within the hole, then the point is classified as not being included in the region.

## SectRegion

Compute the Boolean intersection between two regions.

### Declaration

```
GA_region * NAPI GA_regionFuncs::SectRegion(  
    const GA_region *r1,  
    const GA_region *r2)
```

### Prototype In

snap/graphics.h

### Parameters

<i>r1</i>	First region to compute intersection with
<i>r2</i>	Second region to compute intersection with

### Return Value

Resulting intersection region, or NULL if out of memory.

### Description

Computes the Boolean intersection of two regions, returning the result in a new region. The region may actually be an empty region, in which case the bounding rectangle for the region will be an empty rectangle.

### See Also

*DiffRegion*, *UnionRegion*, *SectRegionRect*

## SectRegionRect

Compute the Boolean intersection between a region and a rectangle.

### Declaration

```
GA_region * NAPI GA_regionFuncs::SectRegionRect(  
    const GA_region *r1,  
    const GA_rect *r2)
```

### Prototype In

snap/graphics.h

### Parameters

<i>r1</i>	Region to compute intersection with
<i>r2</i>	Rectangle to compute intersection with

### Return Value

Resulting intersection region, or NULL if out of memory.

### Description

Computes the Boolean intersection of a region and a rectangle, returning the result in a new region. The region may actually be an empty region, in which case the bounding rectangle for the region will be an empty rectangle. Note that this routine will compute the intersection faster than calling *SectRegion* with a simple region as the second region to intersect.

### See Also

*SectRegion*, *DiffRegion*, *UnionRegion*

## TraverseRegion

Traverses a region for all rectangles in definition.

### Declaration

```
void NAPI GA_regionFuncs::TraverseRegion(
    GA_region *rgn,
    GA_regionCallback doRect,
    void *parms)
typedef void (NAPI GA_regionCallback) (const GA_rect *r, void *parms)
```

### Prototype In

snap/graphics.h

### Parameters

<i>rgn</i>	Region to traverse
<i>doRect</i>	Callback function to call for every rectangle processed
<i>parms</i>	Parameters to pass to callback function

### Description

This function traverses the definition of the region, calling the supplied callback function once for every rectangle in union of rectangles that make up the complex region.

### See Also

*DiffRegion, UnionRegion, SectRegion*

## UnionRegion

Computes the Boolean union of two regions.

### Declaration

```
ibool NAPI GA_regionFuncs::UnionRegion(  
    GA_region *r1,  
    const GA_region *r2)
```

### Prototype In

snap/graphics.h

### Parameters

- |           |  |
|-----------|--|
| <i>r1</i> | Region with which r2 is unioned, and also becomes the result region. |
| <i>r2</i> | Region to be unioned with r1   |

### Return Value

True if the union is valid, false if an empty region was created.

### Description

Computes the Boolean union of two regions for the area covered by region r1 and region r2, computing the resulting region in r1, which may result in an empty region. If you need to retain the value of r1, you need to first copy r1 to a temporary region.

### See Also

*DiffRegion, SectRegion, UnionRegionOfs, UnionRegionRect*



## UnionRegionOfs

Computes the Boolean union of two regions and offsets the result.

### Declaration

```
ibool NAPI GA_regionFuncs::UnionRegionOfs(
    GA_region *r1,
    const GA_region *r2,
    N_int32 dx,
    N_int32 dy)
```

### Prototype In

snap/graphics.h

### Parameters

<i>r1</i>	Region with which r2 is unioned, and also becomes the result region.
<i>r2</i>	Region to be unioned with r1
<i>dx</i>	Offset to add to all x coordinates in region 2
<i>dy</i>	Offset to add to all y coordinates in region 2

### Return Value

True if the union is valid, false if an empty region was created.

### Description

Computes the Boolean union of two regions for the area covered by region r1 and region r2, computing the resulting region in r1, which may result in an empty region. This routine also adds the specified (x,y) offset value to all the coordinates in region 2 before they are unioned with region 1, which allows you to quickly do a union with a translated region without needing to explicitly translate the region itself. If you need to retain the value of r1, you need to first copy r1 to a temporary region.

### See Also

*DiffRegion, SectRegion, UnionRegion, UnionRegionRect*

## UnionRegionRect

Computes the Boolean union of a region and a rectangle.

### Declaration

```
ibool NAPI GA_regionFuncs::UnionRegionRect(  
    GA_region *r1,  
    const GA_rect *r2)
```

### Prototype In

snap/graphics.h

### Parameters

- |           |  |
|-----------|--|
| <i>r1</i> | Region with which r2 is unioned, and also becomes the result region. |
| <i>r2</i> | Rectangle to be unioned with r1                                      |

### Return Value

True if the union is valid, false if an empty region was created.

### Description

Computes the Boolean union of a region *r1* and rectangle *r2*, computing the resulting region in *r1*, which may result in an empty region. If you need to retain the value of *r1*, you need to first copy *r1* to a temporary region.

This routine is faster than using *UnionRegion* if the region to be unioned is a simple rectangle rather than a complex region.

### See Also

*DiffRegion*, *SectRegion*, *UnionRegion*, *UnionRegionOfs*

## GA\_rop3CodesType

### Declaration

```
typedef enum {
    GA_R3_0,
    GA_R3_DPSoon,
    GA_R3_DPSona,
    GA_R3_PSon,
    GA_R3_SDPona,
    GA_R3_DPon,
    GA_R3_PDSxnon,
    GA_R3_PDSaon,
    GA_R3_SDPnaa,
    GA_R3_PDSxon,
    GA_R3_DPna,
    GA_R3_PSDnaon,
    GA_R3_SPna,
    GA_R3_PDSnaon,
    GA_R3_PDSonon,
    GA_R3_Pn,
    GA_R3_PDSona,
    GA_R3_DSon,
    GA_R3_SDPxnon,
    GA_R3_SDPaon,
    GA_R3_DPSxnon,
    GA_R3_DPSaon,
    GA_R3_PSDPSanaxx,
    GA_R3_SSPxDSxaxn,
    GA_R3_SPxPDxa,
    GA_R3_SDPSanaxn,
    GA_R3_PDSPaox,
    GA_R3_SDPxaxn,
    GA_R3_PSDPaox,
    GA_R3_DSPDxaxn,
    GA_R3_PDSox,
    GA_R3_PDSoan,
    GA_R3_DPSnaa,
    GA_R3_SDPxon,
    GA_R3_DSna,
    GA_R3_SPDnaon,
    GA_R3_SPxDSxa,
    GA_R3_PDSPanaxn,
    GA_R3_SDPsaox,
    GA_R3_SDPxnox,
    GA_R3_DPSxa,
    GA_R3_PSDPSaox,
    GA_R3_DPSana,
    GA_R3_SSPxPDxaxn,
    GA_R3_SPDSoax,
    GA_R3_PSDnox,
    GA_R3_PSDPxox,
    GA_R3_PSDnoan,
    GA_R3_PSna,
    GA_R3_SDPnaon,
    GA_R3_SDPSoox,
    GA_R3_Sn,
    GA_R3_SPDSaox,
    GA_R3_SPDSxnox,
    GA_R3_SDPox,
    GA_R3_SDPoan,
    GA_R3_PSDPoax,
```

GA\_R3\_SPDnox,  
 GA\_R3\_SPDSxox,  
 GA\_R3\_SPDnoan,  
 GA\_R3\_Psx,  
 GA\_R3\_SPDSonox,  
 GA\_R3\_SPDSnaox,  
 GA\_R3\_PSan,  
 GA\_R3\_PSDnaa,  
 GA\_R3\_DPSxon,  
 GA\_R3\_SDxPDxa,  
 GA\_R3\_SPDSanaxn,  
 GA\_R3\_SDna,  
 GA\_R3\_DPSnaon,  
 GA\_R3\_DSPDaiox,  
 GA\_R3\_PSDPxaxn,  
 GA\_R3\_SDPxa,  
 GA\_R3\_PDSPDaioxn,  
 GA\_R3\_DPSDoax,  
 GA\_R3\_PDSnox,  
 GA\_R3\_SDPana,  
 GA\_R3\_SSPxDSxoxn,  
 GA\_R3\_PDSPxox,  
 GA\_R3\_PDSnoan,  
 GA\_R3\_PDna,  
 GA\_R3\_DSPnaon,  
 GA\_R3\_DPSDaiox,  
 GA\_R3\_SPDSxaxn,  
 GA\_R3\_DPSonon,  
 GA\_R3\_Dn,  
 GA\_R3\_DPSox,  
 GA\_R3\_DPSoan,  
 GA\_R3\_PDSPoax,  
 GA\_R3\_DPSnox,  
 GA\_R3\_DPx,  
 GA\_R3\_DPSDonox,  
 GA\_R3\_DPSDxox,  
 GA\_R3\_DPSnoan,  
 GA\_R3\_DPSDnaox,  
 GA\_R3\_DPan,  
 GA\_R3\_PDSxa,  
 GA\_R3\_DSPDSaioxn,  
 GA\_R3\_DSPDoax,  
 GA\_R3\_SDPnox,  
 GA\_R3\_SDPSoax,  
 GA\_R3\_DSPnox,  
 GA\_R3\_DSx,  
 GA\_R3\_SDPSONox,  
 GA\_R3\_DSPDSonioxn,  
 GA\_R3\_PDSxxn,  
 GA\_R3\_DPSax,  
 GA\_R3\_PSDPSaioxn,  
 GA\_R3\_SDPax,  
 GA\_R3\_PDSPDaioxn,  
 GA\_R3\_SDPSoax,  
 GA\_R3\_PDSxnan,  
 GA\_R3\_PDSana,  
 GA\_R3\_SSDxPDxaxn,  
 GA\_R3\_SDPSoxox,  
 GA\_R3\_SDPnoan,  
 GA\_R3\_DSPDxox,  
 GA\_R3\_DSPnoan,  
 GA\_R3\_SDPSoax,  
 GA\_R3\_DSan,

GA\_R3\_PDSax,  
 GA\_R3\_DSPDSoaxxn,  
 GA\_R3\_DSPDnoax,  
 GA\_R3\_SDPxnan,  
 GA\_R3\_SPDSnoax,  
 GA\_R3\_DPSxnan,  
 GA\_R3\_SPxDSxo,  
 GA\_R3\_DPSaan,  
 GA\_R3\_DPSaa,  
 GA\_R3\_SPxDSxon,  
 GA\_R3\_DPSxna,  
 GA\_R3\_SPDSnoaxn,  
 GA\_R3\_SDPxna,  
 GA\_R3\_PDSPnoaxn,  
 GA\_R3\_DSPDSoaxx,  
 GA\_R3\_PDSaxn,  
 GA\_R3\_DSa,  
 GA\_R3\_SDPsnaoxn,  
 GA\_R3\_DSPnoa,  
 GA\_R3\_DSPDxoxn,  
 GA\_R3\_SDPnoa,  
 GA\_R3\_SDPsxoaxn,  
 GA\_R3\_SSDxPDxax,  
 GA\_R3\_PDSanan,  
 GA\_R3\_PDSxna,  
 GA\_R3\_SDPsnoaxn,  
 GA\_R3\_DSPDPoaxx,  
 GA\_R3\_SPDaxn,  
 GA\_R3\_PSDPSoaxx,  
 GA\_R3\_DPSaxn,  
 GA\_R3\_DPSxx,  
 GA\_R3\_PSDPsonoxx,  
 GA\_R3\_SDPsonoxn,  
 GA\_R3\_DSxn,  
 GA\_R3\_DPSnax,  
 GA\_R3\_SDPSoaxn,  
 GA\_R3\_SPDnax,  
 GA\_R3\_DSPDoaxn,  
 GA\_R3\_DSPDSaoxx,  
 GA\_R3\_PDSxan,  
 GA\_R3\_DPa,  
 GA\_R3\_PDSPnaoxn,  
 GA\_R3\_DPSnoa,  
 GA\_R3\_DSPDxoxn,  
 GA\_R3\_PDSPonoxn,  
 GA\_R3\_PDxn,  
 GA\_R3\_DSPnax,  
 GA\_R3\_PDSPoaxn,  
 GA\_R3\_DPSoa,  
 GA\_R3\_DPSoxn,  
 GA\_R3\_D,  
 GA\_R3\_DSPsono,  
 GA\_R3\_SPDSxax,  
 GA\_R3\_DSPDsoaxn,  
 GA\_R3\_DSPnao,  
 GA\_R3\_DPno,  
 GA\_R3\_PDSnoa,  
 GA\_R3\_PDSPxoaxn,  
 GA\_R3\_SSPxDSxox,  
 GA\_R3\_SDPanan,  
 GA\_R3\_PSDnax,  
 GA\_R3\_DSPDoaxn,  
 GA\_R3\_DSPDPaoxx,

GA\_R3\_SDPxan,  
 GA\_R3\_PSDPxax,  
 GA\_R3\_DSPDaoxn,  
 GA\_R3\_DPSnao,  
 GA\_R3\_DSno,  
 GA\_R3\_SPDSanax,  
 GA\_R3\_SDxPDxan,  
 GA\_R3\_DPSxo,  
 GA\_R3\_DPSano,  
 GA\_R3\_PSa,  
 GA\_R3\_SPDSnaoxn,  
 GA\_R3\_SPDSonoxn,  
 GA\_R3\_PSnx,  
 GA\_R3\_SPDnoa,  
 GA\_R3\_SPDSxoxn,  
 GA\_R3\_SDPnax,  
 GA\_R3\_PSDPoaxn,  
 GA\_R3\_SDPoa,  
 GA\_R3\_SPDoxn,  
 GA\_R3\_DSPDxax,  
 GA\_R3\_SPDSaoxn,  
 GA\_R3\_S,  
 GA\_R3\_SDPono,  
 GA\_R3\_SDPnao,  
 GA\_R3\_SPno,  
 GA\_R3\_PSDnoa,  
 GA\_R3\_PSDPxoxn,  
 GA\_R3\_PDSnax,  
 GA\_R3\_SPDSaoxn,  
 GA\_R3\_SSPxPDxax,  
 GA\_R3\_DPSanan,  
 GA\_R3\_PSDPSaoxx,  
 GA\_R3\_DPSxan,  
 GA\_R3\_PDSPxax,  
 GA\_R3\_SPDSaoxn,  
 GA\_R3\_DSPDanax,  
 GA\_R3\_SPxDSxan,  
 GA\_R3\_SPDnao,  
 GA\_R3\_SDno,  
 GA\_R3\_SDPxo,  
 GA\_R3\_SDPano,  
 GA\_R3\_PDSoa,  
 GA\_R3\_PDSoxn,  
 GA\_R3\_DSPDxax,  
 GA\_R3\_PSDPaoxn,  
 GA\_R3\_SDPSxax,  
 GA\_R3\_PDSPaoxn,  
 GA\_R3\_SDPSanax,  
 GA\_R3\_SPxPDxan,  
 GA\_R3\_SSPxDSxax,  
 GA\_R3\_DSPDSanaxxn,  
 GA\_R3\_DPSao,  
 GA\_R3\_DPSxno,  
 GA\_R3\_SDPao,  
 GA\_R3\_SDPxno,  
 GA\_R3\_DSo,  
 GA\_R3\_SDPnoo,  
 GA\_R3\_P,  
 GA\_R3\_PDSono,  
 GA\_R3\_PDSnao,  
 GA\_R3\_PSnno,  
 GA\_R3\_PSDnao,  
 GA\_R3\_PDno,

```

GA_R3_PDSxo,
GA_R3_PDSano,
GA_R3_PDSao,
GA_R3_PDSxno,
GA_R3_DPo,
GA_R3_DPSnoo,
GA_R3_PSo,
GA_R3_PSDnoo,
GA_R3_DPSoo,
GA_R3_1
} GA_rop3CodesType

```

**Prototype In**

snap/graphics.h

**Description**

Raster Operation codes for accelerated rendering functions that support ternary operations. The set of mix codes is the standard Microsoft ternary Raster Operation (ROP3) codes between three values, a source, pattern and destination. Note that we don't list the codes here for brevity.

---

**Note:** *Some graphics controllers may not support all ROP3 codes due to hardware bugs, so you must use the GetROP3ExceptionTable function to determine the set of ROP3 codes that the hardware does not properly handle.*

---

## GA\_segment

### Declaration

```
struct GA_segment {  
    struct GA_segment    *next;  
    N_int32               x;  
}
```

### Prototype In

snap/graphics.h

### Description

Fundamental type definition representing a segment within a span that forms a complex region. The segments define the X coordinates of the segments that make up the span. Segments are always in groups of two (start and end segment).

### Members

<i>next</i>	Next segment in span
<i>x</i>	X coordinates of this segment



## GA\_span

### Declaration

```
struct GA_span {  
    struct GA_span  *next;  
    GA_segment      *seg;  
    N_int32         y;  
}
```

### Prototype In

snap/graphics.h

### Description

Fundamental type definition representing a span within a complex region. A span is represented as a list of segments that are included in the span.

### Members

<i>next</i>	Next span in region
<i>seg</i>	Index of first segment in span
<i>y</i>	Y coordinate of this span

## GA\_stipple

**Declaration**

```
typedef N_uint32 GA_stipple
```

**Prototype In**

snap/graphics.h

**Description**

Fundamental type definition for a 16-bit line stipple pattern. Note that we define it as a 32-bit value so it will be passed as a 32-bit argument on the stack correctly when calling 32-bit code from a 16-bit segment.

## GA\_trap

### Declaration

```
typedef struct {
    N_uint32    y;
    N_uint32    count;
    N_fix32     x1;
    N_fix32     x2;
    N_fix32     slope1;
    N_fix32     slope2;
} GA_trap
```

### Prototype In

snap/graphics.h

### Description

Parameter block for the 2D DrawTrap function. This structure is used to pass in the information about a trapezoid to be rendered by the hardware to the driver DrawTrap function.

### Members

<i>y</i>	Starting Y coordinate
<i>count</i>	Number of scanlines to draw
<i>x1</i>	Starting X1 coordinate in 16.16 fixed point
<i>x2</i>	Starting X2 coordinate in 16.16 fixed point
<i>slope1</i>	First edge slope in 16.16 fixed point format
<i>slope2</i>	Second edge slope in 16.16 fixed point format

## GA\_videoFuncs

**Prototype In**

snap/graphics.h

**Description**

Function group containing all hardware video overlay functions available for the device. These functions are used to create, destroy and manage hardware video overlay buffers used for hardware video playback.

---

**Note:** *Be sure to fill in the dwSize member of this structure when you call GA\_queryFunctions to the correct size of the structure at compile time!*

---

## AllocVideoBuffer

Allocate a video overlay buffer of specific dimensions and pixel format.

### Declaration

```
GA_buf * NAPI GA_videoFuncs::AllocVideoBuffer(
    N_int32 width,
    N_int32 height,
    N_int32 format)
```

### Prototype In

snap/graphics.h

### Parameters

<i>width</i>	Width of the video overlay window in pixels
<i>height</i>	Height of the video overlay window in pixels
<i>format</i>	Pixel format for the input data ( <i>GA_VideoBufferFormatsType</i> )

### Return Value

Pointer to allocated video overlay buffer, NULL on failure.

### Description

This function allocates a hardware video overlay buffer in offscreen video memory with the given dimensions and pixel format. The video overlay window dimensions are used by the driver to determine the stretch factor required to interpolate the input video data to the video output window on the display screen. The format parameter is used to determine the pixel format of the input bitmap data stored in offscreen video memory by the application. The video input formats flags supported are defined by *GA\_VideoBufferFormatsType*. The calling application should check the VideoWindows member of the *GA\_modeInfo* structure for the hardware video overlay window capabilities of the graphics hardware for a specific display mode. This function will fail if the application requests an unsupported format or if the requested features are not available for the device installed in the machine. This function will also fail if the maximum number of video overlay buffers supported by the hardware have already been allocated.

### See Also

*AllocVideoBuffer*, *FreeVideoBuffer*, *SetVideoOutput*, *SetVideoColorKey*, *StartVideoFrame*, *EndVideoFrame*

## EndVideoFrame

End decoding a video frame.

### Declaration

```
void NAPI GA_videoFuncs::EndVideoFrame(  
    GA_buf *videoBuffer)
```

### Prototype In

snap/graphics.h

### Parameters

*videoBuffer*                      Video overlay buffer to use

### Description

This function ends the process of displaying a frame of hardware video data on the screen for the specified video overlay buffer. The calling application calls this function after completing the decoding of the next frame so that the hardware can perform any necessary completion steps to get the next frame of video data to the screen. Some hardware does automatic decoding of the video data on the fly and will simply do nothing for this step. However hardware video can be implemented on some hardware by doing a stretch blit from offscreen video memory with interpolation, and this function will execute the stretch blit to get the data to the screen. Some hardware may also use this function to implement double buffering to ensure smooth, tear free video overlay operation.

### See Also

*AllocVideoBuffer, FreeVideoBuffer, SetVideoOutput, SetVideoColorKey, StartVideoFrame, EndVideoFrame*

## FreeVideoBuffer

Frees a video overlay buffer and associated offscreen video memory

### Declaration

```
void NAPI GA_videoFuncs::FreeVideoBuffer(  
    GA_buf *videoBuffer)
```

### Prototype In

snap/graphics.h

### Parameters

*videoBuffer*                      Video overlay buffer to free

### Description

This function frees a previously allocated hardware video overlay buffer, freeing all internal structures and the offscreen video memory used for the overlay buffer.

### See Also

*AllocVideoBuffer, FreeVideoBuffer, SetVideoOutput, SetVideoColorKey, StartVideoFrame, EndVideoFrame*

## SetVideoColorKey

Set the color key for a specific video overlay buffer.

### Declaration

```
void NAPI GA_videoFuncs::SetVideoColorKey(
    GA_buf *videoBuffer,
    GA_color colorKeyLo,
    GA_color colorKeyHi)
```

### Prototype In

snap/graphics.h

### Parameters

<i>videoBuffer</i>	Video overlay buffer to use
<i>colorKeyLo</i>	Color key low value (key for single color key mode)
<i>colorKeyHi</i>	Color key high value (ignored for single color key mode)

### Description

This function sets the color key (or color key range) for the specified video overlay buffer. If single key color keying is enabled, the the colorKeyLo parameter represents the single color key, otherwise the two keys are used to represent the range of colors for the color key.

### See Also

*AllocVideoBuffer, FreeVideoBuffer, SetVideoOutput, SetVideoColorKey, StartVideoFrame, EndVideoFrame*



## SetVideoOutput

Set the video overlay output window dimensions and pixel format.

### Declaration

```
N_int32 NAPI GA_videoFuncs::SetVideoOutput(
    GA_buf *videoBuffer,
    N_int32 left,
    N_int32 top,
    N_int32 width,
    N_int32 height,
    N_int32 flags)
```

### Prototype In

snap/graphics.h

### Parameters

<i>videoBuffer</i>	Video overlay buffer to use
<i>left</i>	Left pixel coordinate of the output window
<i>top</i>	Top pixel coordinate of the output window
<i>width</i>	Width of the input data in pixels
<i>height</i>	Height of the input data in pixels
<i>flags</i>	Pixel format for the input data ( <i>GA_VideoOutputFlagsType</i> )

### Return Value

0 on success, -1 on failure

### Description

This function sets the video output window dimensions for a particular hardware video overlay buffer. This represents the rectangular region on the display screen where the video data will be displayed. The video output rectangle is used by the driver to determine the stretch factor required to interpolate the input video data in offscreen video memory to the video output window on the display screen.

The flags field modifies the way the output image is displayed, and the values are defined in the *GA\_VideoOutputFlagsType* enumeration (enabling interpolation and color keying etc).

### See Also

*AllocVideoBuffer*, *FreeVideoBuffer*, *SetVideoOutput*, *SetVideoColorKey*, *StartVideoFrame*, *EndVideoFrame*

## StartVideoFrame

Start decoding a video frame.

### Declaration

```
N_uint32 NAPI GA_videoFuncs::StartVideoFrame(  
    GA_buf *videoBuffer)
```

### Prototype In

snap/graphics.h

### Parameters

*videoBuffer*                      Video overlay buffer to use

### Description

This function begins the process of displaying a frame of hardware video data on the screen for the specified video overlay buffer. The calling application calls this function before decoding the next frame in the video so that the hardware can perform any setup or synchronisation steps necessary before the application begins decoding the next frame of video data.

### See Also

*AllocVideoBuffer, FreeVideoBuffer, SetVideoOutput, SetVideoColorKey, StartVideoFrame, EndVideoFrame*

## GA\_videoInf

### Declaration

```
typedef struct {
    N_uint32    dwSize;
    N_uint32    VideoInputFormats;
    N_uint32    VideoOutputFlags;
    N_uint16    VideoMinXScale;
    N_uint16    VideoMinYScale;
    N_uint16    VideoMaxXScale;
    N_uint16    VideoMaxYScale;
} GA_videoInf
```

### Prototype In

snap/graphics.h

### Description

Hardware video window information. There is a single structure for each available hardware video window, and defines the capabilities of that hardware video window. The VideoMinXScale and VideoMinYScale members defines the inverse of minimum scale ratio supported by the hardware in the each direction. For instance if the value is 4, the hardware can only scale video down to a window that is 1/4 the size of the source input video. The VideoMaxXScale and VideoMaxYScale define the maxium scale ratio supported by the hardware. For instance if the value is 4, the hardware can only scale video data up to a window that 4 times the size of the source input video.

### Members

<i>dwSize</i>	Set to size of structure in bytes
<i>VideoInputFormats</i>	Hardware video input format flags
<i>VideoOutputFlags</i>	Hardware video output format capabilities
<i>VideoMinXScale</i>	Minimum X scale factor (1/value)
<i>VideoMinYScale</i>	Minimum Y scale factor (1/value)
<i>VideoMaxXScale</i>	Maximum X scale factor
<i>VideoMaxYScale</i>	Maximum Y scale factor

## MCS\_controlsType

### Declaration

```
typedef enum {
    MCS_brightness           = 0x10,
    MCS_contrast             = 0x12,
    MCS_redVideoGain         = 0x16,
    MCS_greenVideoGain       = 0x18,
    MCS_blueVideoGain        = 0x1A,
    MCS_redVideoBlackLevel   = 0x6C,
    MCS_greenVideoBlackLevel = 0x6E,
    MCS_blueVideoBlackLevel  = 0x70,
    MCS_focus                = 0x1C,
    MCS_horizontalPosition    = 0x20,
    MCS_horizontalSize        = 0x22,
    MCS_horizontalPincushion  = 0x24,
    MCS_horizontalPincushionBalance = 0x26,
    MCS_horizontalMisconvergence = 0x28,
    MCS_horizontalLinearity   = 0x2A,
    MCS_horizontalLinearityBalance = 0x2C,
    MCS_verticalPosition      = 0x30,
    MCS_verticalSize          = 0x32,
    MCS_verticalPincushion    = 0x34,
    MCS_verticalPincushionBalance = 0x36,
    MCS_verticalMisconvergence = 0x38,
    MCS_verticalLinearity     = 0x3A,
    MCS_verticalLinearityBalance = 0x3C,
    MCS_parallelogramDistortion = 0x40,
    MCS_trapezoidalDistortion = 0x42,
    MCS_tilt                  = 0x44,
    MCS_topCornerDistortionControl = 0x46,
    MCS_topCornerDistortionBalance = 0x48,
    MCS_bottomCornerDistortionControl = 0x4A,
    MCS_bottomCornerDistortionBalance = 0x4C,
    MCS_hue                   = 0x50,
    MCS_saturation            = 0x52,
    MCS_colorCurveAdjust      = 0x54,
    MCS_horizontalMoire       = 0x56,
    MCS_verticalMoire         = 0x58,
    MCS_audioSpeakerVolume    = 0x62,
    MCS_microphoneSpeakerVolume = 0x64,
    MCS_horAdd                = 0x72,
    MCS_verAdd                = 0x74,
    MCS_bufferAdd             = 0x76,
    MCS_update                = 0x78,
    MCS_adjustFocalPlane      = 0x7A,
    MCS_adjustZoom            = 0x7C,
    MCS_trapezoid             = 0x7E,
    MCS_keystone              = 0x80,
    MCS_horFlip               = 0x82,
    MCS_vertFlip              = 0x84,
    MCS_displayScaling        = 0x86,
    MCS_velocityScanModulation = 0x88,
    MCS_tvColorSaturation     = 0x8A,
    MCS_tvSharpness           = 0x8C,
    MCS_tvContrast            = 0x8E,
    MCS_tvHue                 = 0x90,
    MCS_tvBlackLevel          = 0x92,

    MCS_selectColorPreset     = 0x14,
    MCS_inputLevelSelect1     = 0x5E,
```

```

MCS_inputLevelSelect2      = 0xCC,
MCS_inputSourceSelect1     = 0x60,
MCS_inputSourceSelect2     = 0xCE,
MCS_outputSourceSelect1    = 0xD0,
MCS_outputSourceSelect2    = 0xD2,
MCS_onScreenDisplayEnable  = 0x66,
MCS_onScreenDisplay        = 0xCA,
MCS_languageSelect         = 0x68,
MCS_stereoMode             = 0xD4,
MCS_displayPowerMode       = 0xD6,
MCS_presetColorTemp        = 0xD8,
MCS_scanFormat             = 0xDA,
MCS_displayMode            = 0xDC,
MCS_operationMode          = 0xDE,

MCS_autoSizeCenter         = 0xA2,
MCS_polarityHorizontalSync = 0xA4,
MCS_polarityVerticalSync   = 0xA6,
MCS_syncType               = 0xA8,
MCS_screenOrientation      = 0xAA,
MCS_horFrequency           = 0xAC,
MCS_vertFrequency          = 0xAE,

MCS_degauss                = 0x00
} MCS_controlsType

```

**Prototype In**

snap/ddc.h

**Description**

This enumeration defines the known Monitor Control Command Set controls that can be used to control a monitor via the DDC2Bi protocol. The monitor controls listed here are defined by the VESA Monitor Control Command Set V1.0 specification. Please consult the VESA specification (V1.0 or later) for more information.

## MCS\_polarityFlagsType

### Declaration

```
typedef enum {  
    MCS_vSyncPositive    = 0x01,  
    MCS_hSyncPositive    = 0x02  
} MCS_polarityFlagsType
```

### Prototype In

snap/ddc.h

### Description

This enumeration defines the flags returned by the *MCS\_getTimingReport* function.

### Members

<i>MCS_vSyncPositive</i>	Indicates that vertical sync is positive
<i>MCS_hSyncPositive</i>	Indicates that horizontal sync is positive

## MDBX\_errCodes

### Declaration

```
typedef enum {
    MDBX_ok,
    MDBX_fileNotFound,
    MDBX_corrupt,
    MDBX_outOfMemory,
    MDBX_notFound,
    MDBX_IOError,
    MDBX_parseError,
    MDBX_invalidDB,
    MDBX_dbNotOpen,
    MDBX_noRecords
} MDBX_errCodes
```

### Prototype In

snap/monitor.h

### Description

Error codes returned by monitor database routines

### Members

<i>MDBX_ok</i>	No error
<i>MDBX_fileNotFound</i>	Database file not found
<i>MDBX_corrupt</i>	Database is corrupted
<i>MDBX_outOfMemory</i>	Not enough memory to load DB
<i>MDBX_notFound</i>	Entry was not found in DB
<i>MDBX_IOError</i>	Fatal I/O error
<i>MDBX_parseError</i>	Error parsing monitor database
<i>MDBX_invalidDB</i>	Database handle is invalid
<i>MDBX_dbNotOpen</i>	Attempted to access the database before opening it

## N\_errorType

### Declaration

```
typedef enum {
    nOK,
    nNotDetected,
    nNotPOSTed,
    nDriverNotFound,
    nCorruptDriver,
    nLoadMem,
    nOldVersion,
    nMemMapError,
    nIOError,
    nIRQHookFailed,
    nNotCertified,
    nInternalError,
    nOutOfMemory,
    nOutOfResources,
    nInvalidParameter,
    nNoAGPSupport,
    nInvalidLicense,
    nNotLicensed,
    nMaxError
} N_errorType
```

### Prototype In

snap/common.h

### Description

Error codes returned by N\_status to indicate the driver load status if loading the device driver failed.

### Members

<i>nOK</i>	No error
<i>nNotDetected</i>	Hardware not detected
<i>nNotPOSTed</i>	Hardware has not been POSTed
<i>nDriverNotFound</i>	Driver file not found
<i>nCorruptDriver</i>	File loaded not a driver file
<i>nLoadMem</i>	Not enough memory to load driver
<i>nOldVersion</i>	Driver file is an older version
<i>nMemMapError</i>	Could not map physical memory areas
<i>nIOError</i>	General I/O error
<i>nIRQHookFailed</i>	Could not hook required hardware IRQ
<i>nNotCertified</i>	Driver is not certified
<i>nInternalError</i>	Internal device driver error
<i>nOutOfMemory</i>	Not enough memory to complete operation
<i>nOutOfResources</i>	Not enough spare resources to complete operation
<i>nInvalidParameter</i>	Invalid parameter passed to function
<i>nNoAGPSupport</i>	No AGP support services found
<i>nInvalidLicense</i>	License is invalid
<i>nNotLicensed</i>	Feature is not licensed



## N\_fix32

**Declaration**

```
typedef long N_fix32
```

**Prototype In**

snap/common.h

**Description**

Fundamental type definition for a 32-bit fixed point value. The fixed point value is interpreted as a 16.16 fixed point number, with 16 integral bits and 16 fractional bits.

## Nflt32

**Declaration**

```
typedef float Nflt32
```

**Prototype In**

snap/common.h

**Description**

Fundamental type definition for a 32-bit floating point number. The number is stored as an IEEE 754 floating point number with 1 sign bit, 8 exponent bits and 23 mantissa bits.

## N\_int16

**Declaration**

```
typedef short N_int16
```

**Prototype In**

snap/common.h

**Description**

Fundamental type definition for a 16-bit signed value.

## N\_int32

**Declaration**

```
typedef long N_int32
```

**Prototype In**

snap/common.h

**Description**

Fundamental type definition for a 32-bit signed value.

## N\_int8

**Declaration**

```
typedef char N_int8
```

**Prototype In**

snap/common.h

**Description**

Fundamental type definition for an 8-bit signed value.

## N\_physAddr

**Declaration**

```
typedef unsigned long N_physAddr
```

**Prototype In**

snap/common.h

**Description**

Fundamental type definition for a system physical address

## N\_uint16

**Declaration**

```
typedef unsigned short N_uint16
```

**Prototype In**

snap/common.h

**Description**

Fundamental type definition for a 16-bit unsigned value.

## N\_uint32

**Declaration**

```
typedef unsigned long N_uint32
```

**Prototype In**

snap/common.h

**Description**

Fundamental type definition for a 32-bit unsigned value.



## N\_uint8

**Declaration**

```
typedef unsigned char N_uint8
```

**Prototype In**

snap/common.h

**Description**

Fundamental type definition for an 8-bit unsigned value.

## PE\_errorCodes

### Declaration

```
typedef enum {
    PE_ok,
    PE_fileNotFound,
    PE_outOfMemory,
    PE_invalidDLLImage,
    PE_unableToInitLibC,
    PE_unknownImageFormat
} PE_errorCodes
```

### Prototype In

drvlib/peloder.h

### Description

Defines the error codes returned by the library

### Members

<i>PE_ok</i>	No error
<i>PE_fileNotFound</i>	DLL file not found
<i>PE_outOfMemory</i>	Out of memory loading DLL
<i>PE_invalidDLLImage</i>	DLL image is invalid or corrupted
<i>PE_unableToInitLibC</i>	Unable to initialise the C runtime library
<i>PE_unknownImageFormat</i>	DLL image is in a format that is not supported

## REF2D\_driver

**Prototype In**

snap/graphics.h

**Description**

Structure for the 2d referster rasteriser, which is returned by the *REF2D\_loadDriver* function. This structure also contains all the function pointers used to communicate with the 2d reference rasteriser library.

---

**Note:** *Be sure to fill in the `dwSize` member of this structure when you call `GA_queryFunctions` to the correct size of the structure at compile time!*

---

## DrawRectExtSW

Draws a rectangle entirely in software to a memory buffer

### Declaration

```
void NAPI_REF2D_driver::DrawRectExtSW(
    void *buffer,
    N_int32 dstPitch,
    N_int32 left,
    N_int32 top,
    N_int32 width,
    N_int32 height,
    GA_color color,
    N_int32 mix)
```

### Prototype In

snap/ref2d.h

### Parameters

<i>buffer</i>	Address of buffer to draw into
<i>dstPitch</i>	Scanline pitch of buffer to draw into
<i>left</i>	Left coordinate of the rectangle to draw
<i>top</i>	Top coordinate of the rectangle to draw
<i>width</i>	Width of the rectangle in pixels
<i>height</i>	Height of the rectangle in scanlines
<i>color</i>	Color to fill the rectangle with
<i>mix</i>	Mix to use for drawing the rectangle

### Description

This function is provided to allow the application or shell driver to quickly draw a rectangle in a specific color and mix, anywhere in system memory. The drawing is always done in software, and the buffer can have any starting address and pitch, but all drawing is done in the currently active color depth for the reference rasteriser in use.

## ForceSoftwareOnly

Force software rendering on or off

### Declaration

```
N_int32 NAPI_REF2D_driver::ForceSoftwareOnly(  
    N_int32 enable)
```

### Prototype In

snap/ref2d.h

### Parameters

*enable*                      True to enable software rendering, false if not

### Description

This function is provided to allow the application or shell driver to quickly enable or disable software only rendering. When software only rendering is disabled, a combination of hardware and software functions is used for best performance. When software only rendering is forced, all rendering is done entirely in software.

This is most useful as an application or shell driver debugging aid, to help determine if a problem is hardware specific or not.

## PostSwitchPhysicalResolution

Fix up the software rasteriser after a mode switch has occurred

### Declaration

```
void NAPI_REF2D_driver::PostSwitchPhysicalResolution(void)
```

### Prototype In

snap/ref2d.h

### Description

This function is used to re-initialise the internals for the reference rasteriser library after a physical mode switch has occurred (ie: *PerformDisplaySwitch* after *PollForDisplaySwitch* returned true. Mostly this entails fixing up the internal software renderer to handle changes in hardware functions, such as hardware cursor support etc for the new mode. Application and shell drivers must *always* call this function after calling *PerformDisplaySwitch*.

### See Also

*PerformDisplaySwitch*

## QueryFunctions

Returns the function pointers for the specified ref2d function group.

### Declaration

```
ibool NAPI REF2D_driver::QueryFunctions(
    N_uint32 id,
    void *funcs)
```

### Prototype In

snap/ref2d.h

### Parameters

<i>id</i>	Identifier for the function group to get pointers for
<i>funcs</i>	Pointer to function block to fill in

### Return Value

True if the requested function group is available, false if not.

### Description

This function is the similar to *GA\_queryFunctions* function except that the functions are queried via the 2d reference rasteriser library. This is the function that application level code should use to get access to the SNAP rendering functions that are fleshed out with software rendered functions as necessary.

---

**Note:** *Application code should not call this function directly, but instead call REF2D\_queryFunctions.*

**Note:** *To allow for future compatibility, all function blocks begin with a dwSize member. The caller is expected to fill in the dwSize member with the size of the function block being retrieved before calling QueryFunctions. If the driver exports more functions than the application knows about, only a subset of the functions are copied to the application. If the application expects more functions than the driver provides, the non-existent functions are set to NULL pointers by QueryFunctions, and the remainder copies from the driver.*

---

### See Also

REF2D\_queryFunctions

## RotateBitmap

Sets the software rasteriser active drawing buffer.

### Declaration

```
void * NAPI REF2D_driver::RotateBitmap(
    void *src,
    N_int32 bitsPerPixel,
    N_int32 *stride,
    N_int32 width,
    N_int32 height)
```

### Prototype In

snap/ref2d.h

### Parameters

<i>src</i>	Pointer to the source bitmap to rotate
<i>bitsPerPixel</i>	Pixel depth of the source bitmap in bits
<i>stride</i>	Stride of the source bitmap in bytes
<i>width</i>	Width of the source bitmap
<i>height</i>	Height of the source bitmap

### Return Value

Pointer to the rotated bitmap image in video memory

### Description

This optional function, when implemented, will perform any necessary rotation or transformations on the source bitmap in as needed by the active filter drivers (ie: rotation, flipped, multi-controller etc). The resulting rotated or transformed bitmap will be stored in offscreen video memory, and a pointer to it returned to the caller.



## SetColorCompareMask

Force the software rasteriser color compare mask

### Declaration

```
N_uint32 NAPI_REF2D_driver::SetColorCompareMask(  
    N_uint32 mask)
```

### Prototype In

snap/ref2d.h

### Parameters

*mask*                      Color compare mask to use

### Description

This function is used to override the internal software rendering color compare mask used for source and destination transparent blit functions. If this mask is set to 0xFFFFFFFF, compare compare masking is disabled (the default state). Otherwise it is enabled. This is primarily used by the conformance tests to ensure that color compares are done with the appropriate color mask, while regular application and shell driver code does not do this for speed reasons.

## SetDrawBuffer

Sets the software rasteriser active drawing buffer.

### Declaration

```
N_int32 NAPI_REF2D_driver::SetDrawBuffer(
    GA_buffer *drawBuf,
    void *framebuffer,
    N_int32 bitsPerPixel,
    GA_pixelFormat *pf,
    GA_devCtx *hwCtx,
    N_int32 softwareOnly)
```

### Prototype In

snap/ref2d.h

### Parameters

<i>drawBuf</i>	Buffer to make the active drawing buffer
<i>framebuffer</i>	Pointer to the start of framebuffer memory
<i>bitsPerPixel</i>	Color depth for the buffer
<i>pf</i>	Pixel format for the buffer
<i>hwCtx</i>	SNAP driver to use (NULL if none)
<i>softwareOnly</i>	True to force software only mode

### Return Value

0 on success, -1 on failure.

### Description

This function allows the application to make a video memory or system memory buffer the active rendering buffer for all subsequent drawing commands for the 2d reference rasteriser. This function mimics *SetDrawBuffer*, however it also allows the software rasteriser to be pointed at a system memory buffer for drawing without any hardware acceleration at all.

If the *hwCtx* parameter is set to NULL, this function will enable only software rendering to a system memory buffer pointed to by the *frameBuffer* pointer. If the *hwCtx* member is not NULL and the *frameBuffer* pointer points to a located in video memory, hardware acceleration will be enabled when possible (provided *softwareOnly* is also set to false). If the *softwareOnly* flag is true, no hardware rendering will be used at all.

### See Also

*SetDrawSurface*, *SetDrawBuffer*

## SetDrawSurface

Sets the software rasteriser active drawing buffer.

### Declaration

```
void NAPI_REF2D_driver::SetDrawSurface(
    void *surface,
    N_int32 xRes,
    N_int32 yRes,
    N_int32 bytesPerLine,
    N_int32 bitsPerPixel,
    GA_pixelFormat *pf)
```

### Prototype In

snap/ref2d.h

### Parameters

<i>surface</i>	Pointer to the start of the memory buffer for drawing
<i>xRes</i>	X resolution for the surface
<i>yRes</i>	Y resolution for the surface
<i>bytesPerLine</i>	Scanline width for the memory buffer for drawing
<i>bitsPerPixel</i>	New color depth for the surface
<i>pf</i>	New pixel format for the surface

### Description

This function changes the software draw buffer quickly, to allow the software renderer to render to bitmaps in different locations in memory. This function is tuned for speed, so that we can avoid the overhead of setting the full draw buffer for the reference rasteriser code.

This function is also used by the buffer manager code to quickly change the draw buffer surface.

### See Also

*SetDrawBuffer*

## *PM Library Reference*

---

This section contains the function and data structure references for the Portability Manager (PM) library used by the SciTech SNAP Graphics Architecture. The PM library provides a suite of functions used by the Binary Portable SNAP drivers, which abstract the Binary Portable driver code from operating system specific functions. There are versions of the PM library for every supported Operating System and target environment that provides the interface layer to between the Binary Portable device drivers and the target Operating System platform. Porting the SciTech SNAP drivers to a new Operating System is simply a matter of developing a new version of the PM library for that Operating System. This reference is intended primarily as a reference for developers porting the PM library to new platforms; application developers should generally not use the PM library functions directly.

## *External Functions*

---

## CPU\_getProcessorName

Returns a string defining the speed and name of the processor.

### Declaration

```
char * ZAPI CPU_getProcessorName(void)
```

### Prototype In

cpuinfo.h

### Return Value

Processor name string.

### Description

This function returns an English string describing the speed and name of the CPU.

### See Also

*CPU\_getProcessorType, CPU\_haveMMX, CPU\_getProcessorName*

## CPU\_getProcessorSpeed

Returns the speed of the processor in MHz.

### Declaration

```
ulong ZAPI CPU_getProcessorSpeed(
    ibool accurate)
```

### Prototype In

cpuinfo.h

### Parameters

*accurate*                      True of the speed should be measured accurately

### Return Value

Processor speed in MHz.

### Description

This function returns the speed of the CPU in MHz. Note that if the speed cannot be determined, this function will return 0.

If the accurate parameter is set to true, this function will spend longer profiling the speed of the CPU, and will not round the CPU speed that is reported. This is important for highly accurate timing using the Pentium RDTSC instruction, but it does take a lot longer for the profiling to produce accurate results.

### See Also

CPU\_getProcessorSpeedInHz, *CPU\_getProcessorType*, *CPU\_haveMMX*,  
*CPU\_getProcessorName*

## CPU\_getProcessorSpeedInHZ

Returns the speed of the processor in Hz.

### Declaration

```
void ZAPI CPU_getProcessorSpeedInHZ(  
    ibool accurate,  
    CPU_largeInteger *speed)
```

### Prototype In

cpuinfo.h

### Return Value

Accurate processor speed in Hz.

### Description

This function returns the accurate speed of the CPU in Hz. Note that if the speed cannot be determined, this function will return 0.

This function is similar to the *CPU\_getProcessorSpeed* function, except that it attempts to accurately measure the CPU speed in Hz. This is used internally in the Zen Timer libraries to provide accurate real world timing information. This is important for highly accurate timing using the Pentium RDTSC instruction, but it does take a lot longer for the profiling to produce accurate results.

### See Also

*CPU\_getProcessorSpeed*, *CPU\_getProcessorType*, *CPU\_haveMMX*, *CPU\_getProcessorName*



## CPU\_getProcessorType

Returns the type of processor in the system.

### Declaration

```
uint ZAPI CPU_getProcessorType(void)
```

### Prototype In

cpuinfo.h

### Return Value

Numerical identifier for the installed processor

### Description

Returns the type of processor in the system. Note that if the CPU is an unknown Pentium family processor that we don't have an enumeration for, the return value will be greater than or equal to the value of CPU\_UnkPentium (depending on the value returned by the CUID instruction).

### See Also

*CPU\_getProcessorSpeed, CPU\_haveMMX, CPU\_getProcessorName*

## CPU\_have3DNow

Returns true if the processor supports AMD 3DNow! extensions.

### Declaration

```
ibool ZAPI CPU_have3DNow(void)
```

### Prototype In

cpuinfo.h

### Return Value

True if 3DNow! is available, false if not.

### Description

This function determines if the processor supports the AMD 3DNow! extended instruction set.

### See Also

*CPU\_getProcessorType, CPU\_getProcessorSpeed, CPU\_haveMMX, CPU\_haveSSE, CPU\_getProcessorName*

## CPU\_haveMMX

Returns true if the processor supports Intel MMX extensions.

### Declaration

```
ibool ZAPI CPU_haveMMX(void)
```

### Prototype In

cpuinfo.h

### Return Value

True if MMX is available, false if not.

### Description

This function determines if the processor supports the Intel MMX extended instruction set.

### See Also

*CPU\_getProcessorType, CPU\_getProcessorSpeed, CPU\_have3DNow, CPU\_haveSSE, CPU\_getProcessorName*

## CPU\_haveRDTSC

Returns true if the processor supports RDTSC extensions.

### Declaration

```
ibool ZAPI CPU_haveRDTSC(void)
```

### Prototype In

cpuinfo.h

### Return Value

True if RTSC is available, false if not.

### Description

This function determines if the processor supports the RDTSC instruction for reading the processor time stamp counter.

### See Also

*CPU\_getProcessorType, CPU\_getProcessorSpeed, CPU\_haveMMX, CPU\_have3DNow, CPU\_getProcessorName*

## CPU\_haveSSE

Returns true if the processor supports Intel SSE extensions.

### Declaration

```
ibool ZAPI CPU_haveSSE(void)
```

### Prototype In

cpuinfo.h

### Return Value

True if Intel SSE is available, false if not.

### Description

This function determines if the processor supports the Intel SSE extended instruction set.

### See Also

*CPU\_getProcessorType, CPU\_getProcessorSpeed, CPU\_haveMMX, CPU\_have3DNow, CPU\_getProcessorName*

## EVT\_allowLEDS

Enables/disables the update of the keyboard LED status indicators.

### Declaration

```
void EVTAPI EVT_allowLEDS(  
    ibool enable)
```

**Prototype In**  
event.h

### Parameters

*enable*                      True to enable, false to disable

### Description

Enables the update of the keyboard LED status indicators. Sometimes it may be convenient in the application to turn off the updating of the LED status indicators (such as if a game is using the CAPSLOCK key for some function). Passing in a value of FALSE to this function will turn off all the LEDS, and stop updating them when the internal status changes (note however that internally we still keep track of the toggle key status!).

## EVT\_asciiCode

Macro to extract the ASCII code from a message.

### Declaration

```
uchar EVT_asciiCode(  
    ulong message)
```

**Prototype In**  
event.h

### Parameters

*message*                      Message to extract ASCII code from

### Return Value

ASCII code extracted from the message.

### Description

Macro to extract the ASCII code from the message field of the *event\_t* structure. You pass the message field to the macro as the parameter and the ASCII code is the result, for example:

```
event_t EVT.myEvent;  
uchar   code;  
code = EVT_asciiCode(EVT.myEvent.message);
```

### See Also

*EVT\_scanCode*, *EVT\_repeatCount*

## EVT\_flush

Flushes all events of a specified type from the event queue.

### Declaration

```
void EVTAPI EVT_flush(  
    ulong mask)
```

### Prototype In

event.h

### Parameters

*mask*                      Mask specifying the types of events that should be removed

### Description

Flushes (removes) all pending events of the specified type from the event queue. You may combine the masks for different event types with a simple logical OR.

### See Also

*EVT\_getNext, EVT\_halt, EVT\_peekNext*



## EVT\_getCodePage

Returns the currently active code page for translation of keyboard characters.

### Declaration

```
codepage_t * EVTAPI EVT_getCodePage(void)
```

### Prototype In

event.h

### Return Value

Pointer to the currently active code page translation table.

### Description

This function is returns a pointer to the currently active code page translation table. See *EVT\_setCodePage* for more information.

### See Also

*EVT\_setCodePage*

## EVT\_getHeartBeatCallback

Returns the current user supplied event heartbeat callback function.

### Declaration

```
void EVTAPI EVT_getHeartBeatCallback(  
    _EVT_heartBeatCallback *callback,  
    void **params)
```

### Prototype In

event.h

### Parameters

<i>callback</i>	Place to store the address of user supplied event heartbeat callback
<i>params</i>	Place to store the parameters to pass to the event heartbeat function

### Description

This function retrieves the current event heartbeat function that gets called every time that *EVT\_getNext* or *EVT\_peekNext* is called.

### See Also

*EVT\_getNext*, *EVT\_peekNext*, *EVT\_setHeartBeatCallback*

## EVT\_getMousePos

Returns the current mouse cursor location.

### Declaration

```
void EVTAPI EVT_getMousePos(  
    int *x,  
    int *y)
```

### Prototype In

event.h

### Parameters

<i>x</i>	Place to store value for mouse x coordinate (screen coordinates)
<i>y</i>	Place to store value for mouse y coordinate (screen coordinates)

### Description

Obtains the current mouse cursor position in screen coordinates. Normally the mouse cursor location is tracked using the mouse movement events that are posted to the event queue when the mouse moves, however this routine provides an alternative method of polling the mouse cursor location.

### See Also

*EVT\_setMousePos*

## EVT\_getNext

Retrieves the next pending event from the event queue.

### Declaration

```
ibool EVTAPI EVT_getNext(  
    event_t *evt,  
    ulong mask)
```

### Prototype In

event.h

### Parameters

<i>evt</i>	Pointer to structure to return the event info in
<i>mask</i>	Mask specifying the types of events that should be removed

### Return Value

True if an event was pending, false if not.

### Description

Retrieves the next pending event from the event queue, and stores it in a *event\_t* structure. The mask parameter is used to specify the type of events to be removed, and can be any logical combination of any of the flags defined by the *EVT\_eventType* enumeration.

The what field of the event contains the event code of the event that was extracted. All application specific events should begin with the EVT\_USEREVT code and build from there. Since the event code is stored in an integer, there is a maximum of 32 different event codes that can be distinguished. You can store extra information about the event in the message field to distinguish between events of the same class (for instance the button used in a EVT\_MOUSEBUTTONDOWN event).

If an event of the specified type was not in the event queue, the what field of the event will be set to NULLEVT, and the return value will return false.

---

**Note:** *You should always use the EVT EVERYEVT mask for extracting events from your main event loop handler. Using a mask for only a specific type of event for long periods of time will cause the event queue to fill up with events of the type you are ignoring, eventually causing the application to hang when the event queue becomes full.*

---

### See Also

*EVT\_flush, EVT\_halt, EVT\_peekNext*

## EVT\_halt

Halts until an event of the specified type is received.

### Declaration

```
void EVTAPI EVT_halt(  
    event_t *evt,  
    ulong mask)
```

### Prototype In

event.h

### Parameters

<i>evt</i>	Pointer to
<i>mask</i>	Mask specifying the types of events that should be removed

### Description

This function halts execution until an event of the specified type is received into the event queue. It does not flush the event queue of events before performing the busy loop. However this function does throw away any events other than the ones you have requested via the event mask, to avoid the event queue filling up with unwanted events (like EVT\_KEYUP or EVT\_MOUSEMOVE events).

### See Also

*EVT\_getNext, EVT\_flush, EVT\_peekNext*

## EVT\_isKeyDown

Determines if a specified key is currently down.

### Declaration

```
ibool EVTAPI EVT_isKeyDown(  
    uchar scanCode)
```

### Prototype In

event.h

### Parameters

*scanCode*                      Scan code to test

### Return Value

True if the specified key is currently held down.

### Description

This function determines if a specified key is currently down at the time that the call is made. You simply need to pass in the scan code of the key that you wish to test, and the MGL will tell you if it is currently down or not. The MGL does this by keeping track of the up and down state of all the keys.

## EVT\_joyIsPresent

Returns the mask indicating what joystick axes are attached.

### Declaration

```
int EVTAPI EVT_joyIsPresent(void)
```

### Prototype In

event.h

### Description

This function is used to detect the attached joysticks, and determine what axes are present and functioning. This function will re-detect any attached joysticks when it is called, so if the user forgot to attach the joystick when the application started, you can call this function to re-detect any newly attached joysticks.

### See Also

*EVT\_joySetLowerRight, EVT\_joySetCenter, EVT\_joyIsPresent*

## EVT\_joySetCenter

Calibrates the joystick center position

### Declaration

```
void EVTAPI EVT_joySetCenter(void)
```

### Prototype In

event.h

### Description

This function can be used to zero in on better joystick calibration factors, which may work better than the default simplistic calibration (which assumes the joystick is centered when the event library is initialised). To use this function, ask the user to hold the stick in the center position and then have them press a key or button. and then call this function. This function will then read the joystick and update the calibration factors.

Usually, assuming that the stick was centered when the event library was initialized, you really only need to call *EVT\_joySetLowerRight* since the upper left position is usually always 0,0 on most joysticks. However, the safest procedure is to call all three calibration functions.

### See Also

*EVT\_joySetUpperLeft*, *EVT\_joySetLowerRight*, *EVT\_joySetCenter*



## EVT\_joySetLowerRight

Calibrates the joystick lower right position

### Declaration

```
void EVTAPI EVT_joySetLowerRight(void)
```

### Prototype In

event.h

### Description

This function can be used to zero in on better joystick calibration factors, which may work better than the default simplistic calibration (which assumes the joystick is centered when the event library is initialised). To use this function, ask the user to hold the stick in the lower right position and then have them press a key or button. and then call this function. This function will then read the joystick and update the calibration factors.

Usually, assuming that the stick was centered when the event library was initialized, you really only need to call *EVT\_joySetLowerRight* since the upper left position is usually always 0,0 on most joysticks. However, the safest procedure is to call all three calibration functions.

### See Also

*EVT\_joySetUpperLeft*, *EVT\_joySetCenter*, *EVT\_joyIsPresent*

## EVT\_joySetUpperLeft

Calibrates the joystick upper left position

### Declaration

```
void EVTAPI EVT_joySetUpperLeft(void)
```

### Prototype In

event.h

### Description

This function can be used to zero in on better joystick calibration factors, which may work better than the default simplistic calibration (which assumes the joystick is centered when the event library is initialised). To use this function, ask the user to hold the stick in the upper left position and then have them press a key or button. and then call this function. This function will then read the joystick and update the calibration factors.

Usually, assuming that the stick was centered when the event library was initialized, you really only need to call *EVT\_joySetLowerRight* since the upper left position is usually always 0,0 on most joysticks. However, the safest procedure is to call all three calibration functions.

### See Also

*EVT\_joySetUpperLeft*, *EVT\_joySetLowerRight*, *EVT\_joyIsPresent*

## EVT\_peekNext

Peeks at the next pending event in the event queue.

### Declaration

```
ibool EVTAPI EVT_peekNext(  
    event_t *evt,  
    ulong mask)
```

### Prototype In

event.h

### Parameters

<i>evt</i>	Pointer to structure to return the event info in
<i>mask</i>	Mask specifying the types of events that should be removed

### Return Value

True if an event is pending, false if not.

### Description

Peeks at the next pending event of the specified type in the event queue. The mask parameter is used to specify the type of events to be peeked at, and can be any logical combination of any of the flags defined by the *EVT\_eventType* enumeration.

In contrast to *EVT\_getNext*, the event is not removed from the event queue. You may combine the masks for different event types with a simple logical OR.

### See Also

*EVT\_flush*, *EVT\_getNext*, *EVT\_halt*

## EVT\_pollJoystick

Polls the joystick for position and button information.

### Declaration

```
void EVTAPI EVT_pollJoystick(void)
```

### Prototype In

event.h

### Description

This routine is used to poll analogue joysticks for button and position information. It should be called once for each main loop of the user application, just before processing all pending events via *EVT\_getNext*. All information polled from the joystick will be posted to the event queue for later retrieval.

---

**Note:** *Most analogue joysticks will provide readings that change even though the joystick has not moved. Hence if you call this routine you will likely get an EVT\_JOYMOVE event every time through your event loop.*

---

### See Also

*EVT\_getNext, EVT\_peekNext, EVT\_joySetUpperLeft, EVT\_joySetLowerRight, EVT\_joySetCenter, EVT\_joyIsPresent*

## EVT\_post

Posts a user defined event to the event queue

### Declaration

```
ibool EVTAPI EVT_post(  
    ulong which,  
    ulong what,  
    ulong message,  
    ulong modifiers)
```

### Prototype In

event.h

### Parameters

<i>which</i>	Information about which window got the event
<i>what</i>	Type code for message to post
<i>message</i>	Event specific message to post
<i>modifiers</i>	Event specific modifier flags to post

### Return Value

True if event was posted, false if event queue is full.

### Description

This routine is used to post user defined events to the event queue.

### See Also

*EVT\_flush, EVT\_getNext, EVT\_peekNext, EVT\_halt*

## EVT\_repeatCount

Macro to extract the repeat count from a message.

### Declaration

```
short EVT_repeatCount(  
    ulong message)
```

**Prototype In**  
event.h

### Parameters

*message*                      Message to extract repeat count from

### Return Value

Repeat count extracted from the message.

### Description

Macro to extract the repeat count from the message field of the event structure. The repeat count is the number of times that the key repeated before there was another keyboard event to be place in the queue, and allows the event handling code to avoid keyboard buffer overflow conditions when a single key is held down by the user. If you are processing a key repeat code, you will probably want to check this field to see how many key repeats you should process for this message.

### See Also

*EVT\_asciiCode, EVT\_repeatCount*

## EVT\_scanCode

Macro to extract the keyboard scan code from a message.

### Declaration

```
uchar EVT_scanCode(  
    ulong message)
```

**Prototype In**  
event.h

### Parameters

*message*                      Message to extract scan code from

### Return Value

Keyboard scan code extracted from the message.

### Description

Macro to extract the keyboard scan code from the message field of the event structure. You pass the message field to the macro as the parameter and the scan code is the result, for example:

```
event_t EVT.myEvent;  
uchar   code;  
code = EVT_scanCode(EVT.myEvent.message);
```

---

**Note:** *Scan codes in the event library are not really hardware scan codes, but rather virtual scan codes as generated by a low level keyboard interface driver. All virtual scan code values are defined by the EVT\_scanCodesType enumeration, and will be identical across all supports OS'es and platforms.*

---

### See Also

*EVT\_asciiCode, EVT\_repeatCount*

## EVT\_setCodePage

Sets the currently active code page for translation of keyboard characters.

### Declaration

```
void EVTAPI EVT_setCodePage(  
    codepage_t *page)
```

**Prototype In**  
event.h

### Parameters

*page*                      New code page to make active

### Description

This function is used to set a new code page translation table that is used to translate virtual scan code values to ASCII characters for different keyboard configurations. The default is usually US English, although if possible the PM library will auto-detect the correct code page translation for the target OS if OS services are available to determine what type of keyboard is currently attached.

### See Also

*EVT\_getCodePage*



## EVT\_setHeartBeatCallback

Installs a user supplied event heartbeat callback function.

### Declaration

```
void EVTAPI EVT_setHeartBeatCallback(  
    _EVT_heartBeatCallback callback,  
    void *params)
```

### Prototype In

event.h

### Parameters

<i>callback</i>	Address of user supplied event heartbeat callback
<i>params</i>	Parameters to pass to the event heartbeat function

### Description

This function allows the application programmer to install an event heartbeat function that gets called every time that *EVT\_getNext* or *EVT\_peekNext* is called. This is primarily useful for simulating text mode cursors inside event handling code when running in graphics modes as opposed to hardware text modes.

### See Also

*EVT\_getNext*, *EVT\_peekNext*, *EVT\_getHeartBeatCallback*

## EVT\_setMousePos

Set the mouse position for the event module

### Declaration

```
void EVTAPI EVT_setMousePos(  
    int x,  
    int y)
```

### Prototype In

event.h

### Parameters

<i>x</i>	X coordinate to move the mouse cursor position to
<i>y</i>	Y coordinate to move the mouse cursor position to

### Description

This function moves the mouse cursor position for the event module to the specified location.

### See Also

*EVT\_getMousePos*

## EVT\_setUserEventFilter

Installs a user supplied event filter callback for event handling.

### Declaration

```
void EVTAPI EVT_setUserEventFilter(
    _EVT_userEventFilter filter)
```

**Prototype In**  
event.h

### Description

This function allows the application programmer to install an event filter callback for event handling. Once you install your callback, the MGL event handling routines will call your callback with a pointer to the new event that will be placed into the event queue. Your callback can the modify the contents of the event before it is placed into the queue (for instance adding custom information or perhaps high precision timing information).

If your callback returns FALSE, the event will be ignore and will not be posted to the event queue. You should always return true from your event callback unless you plan to use the events immediately that they are recieved.

---

**Note:** *Your event callback may be called in response to a hardware interrupt and will be executing in the context of the hardware interrupt handler under MSDOS (ie: keyboard interrupt or mouse interrupt). For this reason the code pages for the callback that you register must be locked in memory with the PM\_lockCodePages function. You must also lock down any data pages that your function needs to reference as well.*

**Note:** *You can also use this filter callback to process events at the time they are activated by the user (ie: when the user hits the key or moves the mouse), but make sure your code runs as fast as possible as it will be executing inside the context of an interrupt handler on some systems.*

---

### See Also

EVT\_getNext, EVT\_peekNext

## LZTimerCount

Returns the current count for the Long Period Zen Timer.

### Declaration

```
ulong ZAPI LZTimerCount(void)
```

### Prototype In

ztimer.h

### Return Value

Count that has elapsed in microseconds.

### Description

Obsolete function. You should use the *LZTimerCountExt* function instead which allows for multiple timers running at the same time.

## LZTimerCountExt

Returns the current count for the Long Period Zen Timer.

### Declaration

```
ulong ZAPI LZTimerCountExt(  
    LZTimerObject *tm)
```

### Prototype In

ztimer.h

### Parameters

*tm*            Timer object to compute the elapsed time with.

### Return Value

Count that has elapsed in microseconds.

### Description

Returns the current count that has elapsed between calls to *LZTimerOn* and *LZTimerOff* in microseconds.

### See Also

*LZTimerOnExt*, *LZTimerOffExt*, *LZTimerLapExt*

## LZTimerLap

Returns the current count for the Long Period Zen Timer and keeps it running.

### Declaration

```
ulong ZAPI LZTimerLap(void)
```

### Prototype In

ztimer.h

### Return Value

Count that has elapsed in microseconds.

### Description

Obsolete function. You should use the *LZTimerLapExt* function instead which allows for multiple timers running at the same time.

## LZTimerLapExt

Returns the current count for the Long Period Zen Timer and keeps it running.

### Declaration

```
ulong ZAPI LZTimerLapExt(  
    LZTimerObject *tm)
```

### Prototype In

ztimer.h

### Parameters

*tm*                  Timer object to do lap timing with

### Return Value

Count that has elapsed in microseconds.

### Description

Returns the current count that has elapsed since the last call to *LZTimerOn* in microseconds. The time continues to run after this function is called so you can call this function repeatedly.

### See Also

*LZTimerOnExt*, *LZTimerOffExt*, *LZTimerCountExt*

## LZTimerOff

Stops the Long Period Zen Timer counting.

### Declaration

```
void ZAPI LZTimerOff(void)
```

### Prototype In

ztimer.h

### Description

Obsolete function. You should use the *LZTimerOffExt* function instead which allows for multiple timers running at the same time.



## LZTimerOffExt

Stops the Long Period Zen Timer counting.

### Declaration

```
void ZAPI LZTimerOffExt(  
    LZTimerObject *tm)
```

### Prototype In

ztimer.h

### Parameters

*tm*            Timer object to stop timing with

### Description

Stops the Long Period Zen Timer counting and latches the count. Once you have stopped the timer you can read the count with *LZTimerCount*. If you need highly accurate timing, you should use the on and off functions rather than the lap function since the lap function does not subtract the overhead of the function calls from the timed count.

### See Also

*LZTimerOnExt*, *LZTimerLapExt*, *LZTimerCountExt*

## LZTimerOn

Starts the Long Period Zen Timer counting.

### Declaration

```
void ZAPI LZTimerOn(void)
```

### Prototype In

ztimer.h

### Description

Obsolete function. You should use the *LZTimerOnExt* function instead which allows for multiple timers running at the same time.

## LZTimerOnExt

Starts the Long Period Zen Timer counting.

### Declaration

```
void ZAPI LZTimerOnExt(
    LZTimerObject *tm)
```

### Prototype In

ztimer.h

### Parameters

*tm*                      Timer object to start timing with

### Description

Starts the Long Period Zen Timer counting. Once you have started the timer, you can stop it with *LZTimerOff* or you can latch the current count with *LZTimerLap*.

The Long Period Zen Timer uses a number of different high precision timing mechanisms to obtain microsecond accurate timings results whenever possible. The following different techniques are used depending on the operating system, runtime environment and CPU on the target machine. If the target system has a Pentium CPU installed which supports the Read Time Stamp Counter instruction (RDTSC), the Zen Timer library will use this to obtain the maximum timing precision available.

Under 32-bit Windows, if the Pentium RDTSC instruction is not available, we first try to use the Win32 QueryPerformanceCounter API, and if that is not available we fall back on the timeGetTime API which is always supported.

Under 32-bit DOS, if the Pentium RDTSC instruction is not available, we then do all timing using the old style 8253 timer chip. The 8253 timer routines provide highly accurate timings results in pure DOS mode, however in a DOS box under Windows or other Operating Systems the virtualization of the timer can produce inaccurate results.

---

**Note:** *Because the Long Period Zen Timer stores the results in a 32-bit unsigned integer, you can only time periods of up to 2<sup>32</sup> microseconds, or about 1hr 20mins. For timing longer periods use the Ultra Long Period Zen Timer.*

---

### See Also

*LZTimerOffExt, LZTimerLapExt, LZTimerCountExt*

## PCI\_accessReg

Function to read/write values to PCI configuration space registers

### Declaration

```
ulong PCIAPI PCI_accessReg(
    int index,
    ulong value,
    int func,
    PCIDeviceInfo *info)
```

### Prototype In

pcilib.h

### Parameters

<i>index</i>	Index of the register to access
<i>value</i>	Value to write to the register for write access
<i>func</i>	Function to implement ( <i>PCIAccessRegFlags</i> )
<i>info</i>	PCI device information block for device to access

### Return Value

The value read from the register for read operations

### Description

This function can be used to read or write, BYTE, WORD and DWORD values to and from PCI configuration space registers. Please refer to the *PCIAccessRegFlags* type for the different operations supported. The PCI device that is accessed is the one described by the *PCIDeviceInfo* structure passed in the info parameter (or more correctly the *PCIslot* value stored within this structure).

### See Also

*PCI\_getNumDevices*, *PCI\_enumerate*, *PCI\_accessReg*, *PCI\_readRegBlock* *PCI\_writeRegBlock*

## PCI\_enumerate

Enumerates all devices on the PCI bus

### Declaration

```
int PCIAPI PCI_enumerate(  
    PCIDeviceInfo info[])
```

### Prototype In

pcilib.h

### Parameters

*info*                      Array of *PCIDeviceInfo* structures to fill in

### Return Value

Number of PCI devices found and enumerated on the PCI bus, 0 if not PCI.

### Description

Function to enumerate all available devices on the PCI bus into an array of configuration information blocks.

### See Also

*PCI\_getNumDevices, PCI\_enumerate, PCI\_accessReg, PCI\_readRegBlock PCI\_writeRegBlock*

## PCI\_getNumDevices

Returns number of devices on the PCI bus

### Declaration

```
int PCIAPI PCI_getNumDevices(void)
```

### Prototype In

pcilib.h

### Return Value

Number of PCI devices found and enumerated on the PCI bus, 0 if not PCI.

### Description

Function to enumerate the number of available devices on the PCI bus and return the number found.

### See Also

*PCI\_getNumDevices, PCI\_enumerate, PCI\_accessReg, PCI\_readRegBlock PCI\_writeRegBlock*

## PCI\_readRegBlock

Function to read a block of PCI configuration space registers

### Declaration

```
void PCI_API PCI_readRegBlock(
    PCIDeviceInfo *info,
    int index,
    void *dst,
    int count)
```

### Prototype In

pcilib.h

### Parameters

<i>info</i>	PCI device information block for device to access
<i>index</i>	Index of register to start reading from
<i>dst</i>	Place to store the values read from configuration space
<i>count</i>	Count of bytes to read from configuration space

### Description

This function is used to read a block of PCI configuration space registers from the configuration space into the passed in data block. This function will properly handle reading non-DWORD aligned data from the configuration space correctly.

### See Also

*PCI\_getNumDevices, PCI\_enumerate, PCI\_accessReg, PCI\_readRegBlock PCI\_writeRegBlock*

## PCI\_writeRegBlock

Function to read a block of PCI configuration space registers

### Declaration

```
void PCIAPI PCI_writeRegBlock(
    PCIDeviceInfo *info,
    int index,
    void *src,
    int count)
```

### Prototype In

pcilib.h

### Parameters

<i>info</i>	PCI device information block for device to access
<i>index</i>	Index of register to start reading from
<i>src</i>	Place to store the values read from configuration space
<i>count</i>	Count of bytes to read from configuration space

### Description

This function is used to write a block of PCI configuration space registers to the configuration space from the passed in data block. This function will properly handle writing non-DWORD aligned data to the configuration space correctly.

### See Also

*PCI\_getNumDevices, PCI\_enumerate, PCI\_accessReg, PCI\_readRegBlock PCI\_writeRegBlock*



## PE\_freeLibrary

Frees a loaded Portable Binary DLL

### Declaration

```
void WINAPI PE_freeLibrary(  
    PE_MODULE *hModule)
```

### Prototype In

drvlib/peloder.h

### Parameters

*hModule*                      Handle to a loaded PE DLL library to free

### Description

This function frees a loaded PE DLL library from memory.

### See Also

*PE\_getProcAddress*, *PE\_loadLibrary*

## PE\_getError

Returns the error code for the last operation

### Declaration

```
int WINAPI PE_getError(void)
```

### Prototype In

drvlib/peloder.h

### Return Value

Error code for the last operation.

### See Also

*PE\_getProcAddress, PE\_loadLibrary*

## PE\_getFileSize

Find the actual size of a PE file image

### Declaration

```
ulong WINAPI PE_getFileSize(  
    FILE *f,  
    ulong startOffset)
```

### Prototype In

drvlib/peloder.h

### Parameters

<i>f</i>	Handle to open file to read driver from
<i>startOffset</i>	Offset to the start of the driver within the file

### Return Value

Size of the DLL file on disk, or -1 on error

### Description

This function scans the headers for a Portable Binary DLL to determine the length of the DLL file on disk.

## PE\_getProcAddress

Gets a function address from a Portable Binary DLL

### Declaration

```
void * WINAPI PE_getProcAddress(  
    PE_MODULE *hModule,  
    const char *szProcName)
```

### Prototype In

drvlib/peloder.h

### Parameters

<i>hModule</i>	Handle to a loaded PE DLL library
<i>szProcName</i>	Name of the function to get the address of

### Return Value

Pointer to the function, or NULL on failure.

### Description

This function searches for the named, exported function in a loaded PE DLL library, and returns the address of the function. If the function is not found in the library, this function return NULL.

### See Also

*PE\_loadLibrary*, *PE\_freeLibrary*

## PE\_loadLibrary

Loads a Portable Binary DLL into memory

### Declaration

```
PE_MODULE * WINAPI PE_loadLibrary(  
    const char *szDLLName,  
    ibool shared)
```

### Prototype In

drvlib/peloder.h

### Parameters

<i>szDLLName</i>	Name of the PE DLL library to load
<i>shared</i>	True to load module into shared memory

### Return Value

Handle to loaded PE DLL, or NULL on failure.

### Description

This function loads a Portable Binary DLL library from disk, relocates the code and returns a handle to the loaded library. This function will only work on DLL's that do not have any imports, since we don't resolve pimport dependencies in this function.

### See Also

*PE\_getProcAddress*, *PE\_freeLibrary*

## PE\_loadLibraryExt

Loads a Portable Binary DLL into memory from an open file

### Declaration

```
PE_MODULE * WINAPI PE_loadLibraryExt(
    FILE *f,
    ULONG startOffset,
    ULONG *size,
    BOOL shared)
```

### Prototype In

drvlib/peloder.h

### Parameters

<i>f</i>	Handle to open file to read driver from
<i>startOffset</i>	Offset to the start of the driver within the file
<i>size</i>	Place to store the size of the driver loaded
<i>shared</i>	True to load module into shared memory

### Return Value

Handle to loaded PE DLL, or NULL on failure.

### Description

This function loads a Portable Binary DLL library from disk, relocates the code and returns a handle to the loaded library. This function is the same as the regular *PE\_loadLibrary* except that it take a handle to an open file and an offset within that file for the DLL to load.

### See Also

*PE\_loadLibrary*, *PE\_getProcAddress*, *PE\_freeLibrary*

## PE\_loadLibraryMGL

Loads a Portable Binary DLL into memory

### Declaration

```
PE_MODULE * WINAPI PE_loadLibraryMGL(
    const char *szDLLName,
    ibool shared)
```

### Prototype In

drvlib/peloder.h

### Parameters

<i>szDLLName</i>	Name of the PE DLL library to load
<i>shared</i>	True to load module into shared memory

### Return Value

Handle to loaded PE DLL, or NULL on failure.

### Description

This function is the same as the regular *PE\_loadLibrary* function, except that it looks for the drivers in the MGL\_ROOT/drivers directory or a /drivers directory relative to the current directory.

### See Also

*PE\_loadLibraryMGL*, *PE\_getProcAddress*, *PE\_freeLibrary*

## PM\_agpCommitPhysical

Commits memory to a range of reserved physical AGP memory addresses

### Declaration

```
ibool PMAPI PM_agpCommitPhysical(
    void *physContext,
    ulong numPages,
    ulong startOffset,
    PM_physAddr *physAddr)
```

### Prototype In

pmapi.h

### Parameters

<i>physContext</i>	Physical AGP context to commit memory for
<i>numPages</i>	Number of pages to be committed
<i>startOffset</i>	Offset in pages into the reserved physical context
<i>physAddr</i>	Returns the physical address of the committed memory

### Return Value

True on success, false on failure.

### Description

This function commits AGP memory into the specified physical context that was previously reserved by a call to *PM\_agpReservePhysical*. You can use the *startOffset* and *numPages* parameters to only commit portions of the reserved memory range at a time.

### See Also

*PM\_agpReservePhysical*, *PM\_agpFreePhysical*



## **PM\_agpExit**

Close down the AGP functions

### **Declaration**

```
void PMAPI PM_agpExit(void)
```

### **Prototype In**

*pmapi.h*

### **Description**

This function closes down the loaded AGP driver.

### **See Also**

*PM\_agpInit*

## PM\_agpFreePhysical

Frees a range of committed physical AGP memory pages

### Declaration

```
ibool PMAPI PM_agpFreePhysical(
    void *physContext,
    ulong numPages,
    ulong startOffset)
```

### Prototype In

pmapi.h

### Parameters

<i>physContext</i>	Physical AGP context to free memory for
<i>numPages</i>	Number of pages to be freed
<i>startOffset</i>	Offset in pages into the reserved physical context

### Return Value

True on success, false on failure.

### Description

This function frees memory previously committed by the *PM\_agpCommitPhysical* function. Note that you can free a portion of a memory range that was previously committed if you wish.

### See Also

*PM\_agpCommitPhysical*

## PM\_agpInit

Initialise the AGP functions

### Declaration

```
ulong PMAPI PM_agpInit(void)
```

### Prototype In

pmapi.h

### Return Value

Size of AGP aperture in MB on success, 0 on failure.

### Description

This function initialises the AGP driver in the system and returns the size of the available AGP aperture in megabytes if an AGP bus is found. If there is no AGP bus or the driver could not be loaded, this function returns 0.

### See Also

*PM\_agpExit, PM\_agpReservePhysical*

## PM\_agpReleasePhysical

Releases a range of reserved physical AGP memory addresses

### Declaration

```
ibool PMAPI PM_agpReleasePhysical(  
    void *physContext)
```

### Prototype In

pmapi.h

### Parameters

*physContext*                      Physical AGP context to release

### Return Value

True on success, false on failure.

### Description

This function releases a range of physical memory addresses on the system bus which the AGP controller will respond to. All committed memory for the physical address range covered by the context will be released.

### See Also

*PM\_agpReservePhysical*

## PM\_agpReservePhysical

Reserves a range of physical addresses of AGP memory

### Declaration

```
ibool PMAPI PM_agpReservePhysical(
    ulong numPages,
    int type,
    void **physContext,
    PM_physAddr *physAddr)
```

### Prototype In

pmapi.h

### Parameters

<i>numPages</i>	Number of memory pages that should be reserved
<i>type</i>	Type of memory to allocate
<i>physContext</i>	Returns the physical context handle for the mapping
<i>physAddr</i>	Returns the physical address for the mapping

### Return Value

True on success, false on failure.

### Description

This function reserves a range of physical memory addresses on the system bus which the AGP controller will respond to. If this function succeeds, the AGP controller can respond to the reserved physical address range on the bus. However you must first call AGP\_commitPhysical to cause this memory to actually be committed for use before it can be accessed.

### See Also

*PM\_agpReleasePhysical*, *PM\_agpCommitPhysical*

## PM\_allocLockedMem

Allocate a block of locked, physical memory for DMA operations.

### Declaration

```
void * PMAPI PM_allocLockedMem(
    uint size,
    ulong *physAddr,
    ibool contiguous,
    ibool below16M)
```

### Prototype In

pmapi.h

### Parameters

<i>size</i>	Size of memory block to allocate
<i>physAddr</i>	Place to return the physical memory address
<i>contiguous</i>	True if the block should be contiguous
<i>below16M</i>	True if the block must be below 16M physical

### Return Value

Linear pointer to memory block, or NULL on failure.

### Description

This function is used to allocate a block of locked, physical memory for use in hardware DMA operations. If the contiguous parameter is set to true, then the memory block requested must be contiguous in physical memory or this function will fail (requesting physically contiguous memory usually does not succeed for large blocks except immediately after the operating system is loaded or during the boot process). If the below16M flag is set to true, then the physical memory block must be allocated below the 16Mb physical memory address (required for old ISA bus sound card DMA buffers for instance). When this function succeeds, it will return a regular C pointer to the allocated memory block.

Note also that the memory block allocated by this function must also be globally shared, such that the linear address returned will be valid in all processes in the system at the same location.

### See Also

*PM\_freeLockedMem*

## PM\_allocPage

Allocates a page aligned and page sized block of memory

### Declaration

```
void * PMAPI PM_allocPage(  
    ibool locked)
```

### Prototype In

pmapi.h

### Parameters

*locked*                      True if the memory should be locked down, false if not

### Return Value

Pointer to the page aligned page of memory allocated, NULL on failure.

### Description

This function is used to allocate a single page sized and page aligned block of memory from the operating system. The memory block may be optionally locked in physical memory if the locked parameter is set to true.

This function is mostly used to allocate pages of physical memory that are used to back the AGP memory regions used by graphics drivers. As such there is presently no requirement for the pages allocated by this function to be globally mapped.

### See Also

*PM\_freePage*

## PM\_allocRealSeg

Allocate a block of real mode memory

### Declaration

```
void * PMAPI PM_allocRealSeg(
    uint size,
    uint *r_seg,
    uint *r_off)
```

### Prototype In

pmapi.h

### Parameters

<i>size</i>	Size of memory block to allocate
<i>r_seg</i>	Place to store real mode segment address of memory block
<i>r_off</i>	Place to store real mode segment offset of memory block

### Return Value

Linear pointer to the real mode memory block, NULL on failure.

### Description

This function is used to allocate a block of real mode memory for communicating with the real mode BIOS. If this function succeeds, the *r\_seg* and *r\_off* parameters will be filled in with the real mode address of the memory block, and the function will return a regular C style linear pointer to the memory block. This is only supported for operating systems that support BIOS access (ie: the *PM\_haveBIOSAccess* function returns true).

### See Also

*PM\_freeRealSeg*, *PM\_mapRealPointer*



## PM\_backslash

Add a file directory separator to the end of the filename.

### Declaration

```
void PMAPI PM_backslash(  
    char *s)
```

### Prototype In

pmapi.h

### Parameters

*s*               String to append the directory separator character to

### Description

This function is a portable way to add a file directory separator to the end of the filename. The separator added will always work on the target platform, and is used extensively by binary portable modules that need to construct path names dynamically that will work properly on the target operating system.

## PM\_blockUntilTimeout

Block until a specific time has elapsed since the last call

### Declaration

```
void PMAPI PM_blockUntilTimeout(  
    ulong milliseconds)
```

### Prototype In

pmapi.h

### Parameters

<i>milliseconds</i>	Number of milliseconds for delay
---------------------	----------------------------------

### Description

This function will block the calling thread or process until the specified number of milliseconds have passed since the *last* call to this function. The first time this function is called, it will return immediately. On subsequent calls it will block until the specified time has elapsed, or it will return immediately if the time has already elapsed.

This function is useful to provide constant time functionality in a program, such as a frame rate limiter for graphics applications etc.

### See Also

*PM\_sleep*

## PM\_callRealMode

Call a real mode far function.

### Declaration

```
void PMAPI PM_callRealMode(
    uint seg,
    uint off,
    RMREGS *in,
    RMSREGS *sregs)
```

### Prototype In

pmapi.h

### Parameters

<i>seg</i>	Real mode segment address of function to call
<i>off</i>	Real mode segment offset of function to call
<i>in</i>	Register block to load before calling interrupt
<i>sregs</i>	Segment register block to load and return values in

### Description

This function is used call a real mode far function, which is used to call the real mode BIOS functions and drivers directly. If you make calls to the real mode BIOS functions or drivers with this function, there is *no* parameter translation at all. Hence you need to translate any real mode memory pointers etc passed into and returned from this function with the *PM\_allocRealSeg* and *PM\_mapRealPointer* functions.

When this function executes the real mode far function, the machine registers will be loaded with the values passed in the 'in' parameter and the segment registers from the 'sregs' parameter. When the function returns, the values in the machine registers will then be saved into the 'out' parameter and the segment registers into the 'sregs' parameter.

This is only supported for operating systems that support BIOS access (ie: the *PM\_haveBIOSAccess* function returns true).

### See Also

*PM\_int86*, *PM\_int86x*, *PM\_allocRealSeg*

## PM\_calloc

Allocate and clear a large memory block.

### Declaration

```
void * PMAPI PM_calloc(  
    size_t nelem,  
    size_t size)
```

### Prototype In

pmapi.h

### Parameters

<i>nelem</i>	number of contiguous size-byte units to allocate
<i>size</i>	size of unit in bytes

### Return Value

Pointer to allocated memory if successful, NULL if out of memory.

### Description

Allocates a block of memory of length (size \* nelem), and clears the allocated area with zeros (0). If you have changed the memory allocation routines with the *PM\_useLocalMalloc* function, then calls to this function will actually make calls to the local memory allocation routines that you have registered.

### See Also

*PM\_malloc*, *PM\_realloc*, *PM\_free*, *PM\_useLocalMalloc*

## PM\_closeConsole

Closes the OS console.

### Declaration

```
void PMAPI PM_closeConsole(  
    PM_HWND hwndConsole)
```

### Prototype In

pmapi.h

### Parameters

*hwndConsole*                      Console window handle to close

### Description

This function closes the OS console, given the console window handle passed back from the *PM\_openConsole* function.

### See Also

*PM\_openConsole*, *PM\_getConsoleStateSize*, *PM\_saveConsoleState*, *PM\_restoreConsoleState*

## PM\_doSuspendApp

Suspends the application by switch back to the OS desktop

### Declaration

```
void PMAPI PM_doSuspendApp(void)
```

### Prototype In

pmapi.h

### Description

This function suspends the application by switching back to the regular OS desktop, allowing normal application code to be processed and then waiting for the application activate command to bring us back to fullscreen mode.

This version only gets called if we have not captured the screen switch in our activate message loops and will occur if the DirectDraw drivers lose a surface for some reason while rendering. This should not normally happen, but it is included just to be sure (it can happen on WinNT/2000/XP if the user hits the Ctrl-Alt-Del key combination). Note that this code will always spin loop, and we cannot disable the spin looping from this version (ie: if the user hits Ctrl-Alt-Del under WinNT/2000 the application main loop will cease to be executed until the user switches back to the application).

## PM\_enableWriteCombine

Enable write combining for a physical memory region

### Declaration

```
ibool PMAPI PM_enableWriteCombine(
    ulong base,
    ulong length,
    uint type);
int PMAPI PM_enableWriteCombine(ulong base,ulong length,uint type)
```

### Prototype In

pmapi.h

### Parameters

<i>base</i>	Physical base address of region to write combine
<i>length</i>	Length of the region to write combine
<i>type</i>	Type of write caching to enable ( <i>PMEnableWriteCombineFlags</i> )

### Description

This function is used to change the write combine caching values for a physical memory region. The type of caching that can be enabled for the region can be one of the *PMEnableWriteCombineFlags* types.

Note that most CPU's only have a very limited number of write combine regions available, so this function must be used as sparingly as possible to ensure the hardware in the system can get the caching that it needs.

## PM\_enumWriteCombine

Enumerates all write combine regions currently enabled for the processor.

### Declaration

```
int PMAPI PM_enumWriteCombine(  
    PM_enumWriteCombine_t callback)
```

### Prototype In

pmapi.h

### Parameters

*callback*                      Function to callback with write combine information

### Return Value

PM\_MTRR\_ERR\_OK on success, otherwise error code.

### Description

This function is used to enumerate all write combine regions currently enabled for the processor.



## PM\_fatalError

Report a fatal error condition and halt the program.

### Declaration

```
void PMAPI PM_fatalError(  
    const char *msg)
```

### Prototype In

pmapi.h

### Parameters

*msg*                      Message to display as the fatal error prior to exit

### Description

This function is a portable method to report a fatal error condition and then halt program execution. It will display the error message using whatever mechanism is appropriate for the operating system. ie: A console text message in DOS, OS/2, Linux etc, or a popup dialog box for a GUI based environment like Windows, PMSHELL or X11.

### See Also

*PM\_setFatalErrorCleanup*

## PM\_findBPD

Function to find a BPD file on the SNAP driver file path

### Declaration

```
ibool PMAPI PM_findBPD(  
    const char *dllname,  
    char *bpdpath)
```

### Prototype In

pmapi.h

### Parameters

<i><b>dllname</b></i>	Name of the Binary Portable DLL to load
<i><b>bpdpath</b></i>	Place to store the actual path to the file

### Return Value

True if found, false if not.

### Description

Finds the location of a specific Binary Portable DLL, by searching all the standard SciTech SNAP driver locations. If the file is found, we cache the SNAP driver location internally and search for all drivers relative to this path for subsequent calls. Hence the first call to this function should be to find a parent BPD file that defines the root of all the BPD files in the installation (ie: graphics.bpd for the SNAP Graphics API).

### See Also

*PM\_setLocalBPDPath*

## PM\_findClose

Function to close the find process

### Declaration

```
void PMAPI PM_findClose(  
    void *handle)
```

### Prototype In

pmapi.h

### Parameters

*handle*                      Handle return from *PM\_findFirstFile*

### Description

This function is used to close the search handle returned by the *PM\_findFirstFile* function.

### See Also

*PM\_findFirstFile*, *PM\_findNextFile*

## PM\_findFirstFile

Function to find the first file matching a search criteria in a directory.

### Declaration

```
void * PMAPI PM_findFirstFile(  
    const char *filename,  
    PM_findData *findData)
```

### Prototype In

pmapi.h

### Parameters

<i>filename</i>	Filename mask to see the search with
<i>findData</i>	Place to return the found file data

### Return Value

Pointer to the find handle created, PM\_FILE\_INVALID if no more files.

### Description

This function is used to find the first file matching a search criteria in a directory. Once you have found the first file, you can then call *PM\_findNextFile* to find the next file matching the same search criteria. When you are done, make sure you call *PM\_findClose* to free the handle returned by this function.

### See Also

*PM\_findNextFile*, *PM\_findClose*

## PM\_findNextFile

Function to find the next file matching a search criteria in a directory.

### Declaration

```
ibool PMAPI PM_findNextFile(  
    void *handle,  
    PM_findData *findData)
```

### Prototype In

pmapi.h

### Parameters

<i>handle</i>	Handle return from <i>PM_findFirstFile</i>
<i>findData</i>	Place to return the found file data

### Return Value

True if another file is found, false if not.

### Description

This function is used to find the next file matching the same search criteria passed to *PM\_findFirstFile*. You can keep calling *PM\_findNextFile* to find each file that matches until this function returns false, indicating there are no more files that match. When you are done, make sure you call *PM\_findClose* to free the handle returned by the *PM\_findFirstFile* function.

### See Also

*PM\_findFirstFile*, *PM\_findClose*

## PM\_flushTLB

Flush the translation lookaside buffer.

### Declaration

```
void PMAPI PM_flushTLB(void)
```

### Prototype In

pmapi.h

### Description

This function is used to flush the translation lookaside buffer.

## PM\_free

Frees a block of memory.

### Declaration

```
void PMAPI PM_free(  
    void *p)
```

### Prototype In

pmapi.h

### Parameters

*p*                  Pointer to memory block to free

### Description

Frees a block of memory previously allocated with either *PM\_malloc*, *PM\_calloc* or *PM\_realloc*.

### See Also

*PM\_malloc*, *PM\_calloc*, *PM\_realloc*, *PM\_useLocalMalloc*

## PM\_freeLibrary

Unload a shared library.

### Declaration

```
void PMAPI PM_freeLibrary(  
    PM_MODULE hModule)
```

### Prototype In

pmapi.h

### Parameters

*hModule*                      Handle to the module to unload

### Description

This function is used to unload a shared library previously loaded with the *PM\_loadLibrary* function.

### See Also

*PM\_loadLibrary*, *PM\_getProcAddress*



## PM\_freeLockedMem

Free a block of locked physical memory.

### Declaration

```
void PMAPI PM_freeLockedMem(
    void *p,
    uint size,
    ibool contiguous)
```

### Prototype In

pmapi.h

### Parameters

<i>p</i>	Pointer to the memory block to free
<i>size</i>	Size of memory block that was allocated
<i>contiguous</i>	True if the block was contiguously allocated

### Description

This function is used to free a block of locked, physical memory previously allocated by the *PM\_allocLockedMem* function. This function must be passed the exact same size and contiguous values that were passed to the *PM\_allocLockedMem* function, or it may produce strange results.

### See Also

*PM\_allocLockedMem*

## PM\_freePage

Free a page aligned and page sized block of memory

### Declaration

```
void PMAPI PM_freePage(  
    void *p)
```

### Prototype In

pmapi.h

### Parameters

*p*            Linear pointer to the page of memory to free

### Description

This function is used to free a page of memory previously allocated with the *PM\_allocPage* function.

### See Also

*PM\_allocPage*

## PM\_freePhysicalAddr

Free a physical address mapping allocated by *PM\_mapPhysicalAddr*.

### Declaration

```
void PMAPI PM_freePhysicalAddr(  
    void *ptr,  
    ulong limit)
```

### Prototype In

pmapi.h

### Parameters

<i>ptr</i>	Linear address of the address to free
<i>limit</i>	Limit for the mapped memory region (length-1)

### Description

This function is used to free an address mapping previously allocated with the *PM\_mapPhysicalAddr* function.

### See Also

*PM\_mapPhysicalAddr*

## PM\_freeRealSeg

Free a block of real mode memory.

### Declaration

```
void PMAPI PM_freeRealSeg(  
    void *mem)
```

### Prototype In

pmapi.h

### Parameters

*mem*                      Pointer to the memory block to free

### Description

This function is used to free a block of real mode memory previously allocated with the *PM\_allocRealSeg* function. This is only supported for operating systems that support BIOS access (ie: the *PM\_haveBIOSAccess* function returns true).

### See Also

*PM\_allocRealSeg*, *PM\_mapRealPointer*

## PM\_freeShared

Frees a block of global shared memory.

### Declaration

```
void PMAPI PM_freeShared(  
    void *ptr)
```

### Prototype In

pmapi.h

### Parameters

*ptr*                      Shared memory block to free

### Description

This function is used to free a block of global shared memory previously allocated with the *PM\_mallocShared* function.

### See Also

*PM\_mallocShared*

## PM\_getA0000Pointer

Return a pointer to 0xA0000 physical VGA graphics framebuffer.

### Declaration

```
void * PMAPI PM_getA0000Pointer(void)
```

### Prototype In

pmapi.h

### Return Value

Pointer to 0xA0000 physical VGA graphics framebuffer.

### Description

This function is used to obtain a pointer to the physical VGA graphics framebuffer which is located at physical address 0xA0000. This is supported on all operating systems.

## PM\_getBIOSPointer

Return a pointer to the real mode BIOS data area.

### Declaration

```
void * PMAPI PM_getBIOSPointer(void)
```

### Prototype In

pmapi.h

### Return Value

Pointer to the real mode BIOS data area

### Description

This function is used to obtain a pointer to the real mode BIOS data area. This is only possible on machines that provide access to the real mode BIOS, so if the *PM\_haveBIOSAccess* function returns false, this function will not return a useful pointer (so don't try to use it!).

## PM\_getBootDrive

Return the drive letter for the boot drive.

### Declaration

```
char PMAPI PM_getBootDrive(void)
```

### Prototype In

pmapi.h

### Return Value

Character representing the operating system boot drive.

### Description

This function is used to obtain the drive letter for the boot drive used by the operating system. This is only valid for operating systems that use driver letters, such as DOS, OS/2 and Windows.



## PM\_getCOMPort

Return the base I/O port for the specified COM port.

### Declaration

```
int PMAPI PM_getCOMPort(  
    int port)
```

### Prototype In

pmapi.h

### Parameters

*port*                    COM port number to get I/O port for

### Return Value

Base I/O port for the specified COM port

### Description

This function is used to determine from the operating system what the base I/O port is for the specified COM port in the system. This is only used presently for software stereo support on supported operating systems (ie: DOS, Windows 9x and Windows NT/2000/XP).

## PM\_getConsoleStateSize

Find the size of the console state buffer.

### Declaration

```
int PMAPI PM_getConsoleStateSize(void)
```

### Prototype In

pmapi.h

### Return Value

Size of the console state save buffer in bytes

### Description

This function returns the size of the console state save buffer in bytes. This buffer can be used to save and restore the state of the OS console.

### See Also

*PM\_openConsole, PM\_saveConsoleState, PM\_setSuspendAppCallback,  
PM\_restoreConsoleState, PM\_closeConsole*

## PM\_getCurrentPath

Return the current operating system path or working directory.

### Declaration

```
char * PMAPI PM_getCurrentPath(  
    char *path,  
    int maxLen)
```

### Prototype In

pmapi.h

### Parameters

<i>path</i>	Place to store the path string
<i>maxLen</i>	Maximum length of the path string

### Return Value

Pointer to the current path string

### Description

This function is used to obtain the current operating system path or working directory. The string is copied into the path parameter, with a maximum length of maxLen characters. A pointer to path is also returned from the function.

### See Also

*PM\_getdcwd*

## PM\_getDirectDrawWindow

Returns a pointer to the DirectDraw window

### Declaration

```
PM_HWND PMAPI PM_getDirectDrawWindow(void)
```

### Prototype In

pmapi.h

### Return Value

Pointer to the DirectDraw application window.

### Description

Return the DirectDraw window handle used by the application. This is used by the SNAP DirectX driver to find the proper window handle registered by the application for the DirectX fullscreen application.

---

**Note:** *This function is Windows specific*

---

### See Also

*PM\_loadDirectDraw, PM\_unloadDirectDraw*

## PM\_getFileAttr

Function to get the file attributes for a specific file.

### Declaration

```
uint PMAPI PM_getFileAttr(  
    const char *filename)
```

### Prototype In

pmapi.h

### Parameters

*filename*                      Full path to filename for file to get attributes from

### Return Value

Current attributes for the file (*PMFileFlagsType*)

### Description

This function is used to retrieve the current file attributes for a specific file.

### See Also

*PM\_setFileAttr*

## PM\_getFileTime

Function to get the file time and date for a specific file.

### Declaration

```
ibool PMAPI PM_getFileTime(  
    const char *filename,  
    ibool gmTime,  
    PM_time *time)
```

### Prototype In

pmapi.h

### Parameters

<i>filename</i>	Full path to filename for file to get date and time from
<i>gmTime</i>	True if time should be in the GMT timezone
<i>time</i>	Place to store the file time for the file

### Return Value

True on success, false on failure.

### Description

This function is used to obtain the file date and time stamp for a specific file. If the gmTime parameter is true, the time is returned in the GMT time zone, otherwise it is in the local machine time zone.

### See Also

*PM\_setFileTime*

## PM\_getIOPL

Get the I/O priveledge level for the process

### Declaration

```
int PMAPI PM_getIOPL(void)
```

### Prototype In

pmapi.h

### Return Value

Current IOPL active for the process

### Description

This function is used to obtain the I/O priveledge level of the current process.

### See Also

*PM\_setIOPL*

## PM\_getLPTPort

Return the base I/O port for the specified printer port.

### Declaration

```
int PMAPI PM_getLPTPort(  
    int port)
```

### Prototype In

pmapi.h

### Parameters

*port*                      Printer port number to get I/O port for

### Return Value

Base I/O port for the specified printer port

### Description

This function is used to determine from the operating system what the base I/O port is for the specified printer port in the system. This is only used presently for software stereo support on supported operating systems (ie: DOS, Windows 9x and Windows NT/2000/XP).



## PM\_getMachineName

Get the name of the machine on the network.

### Declaration

```
const char * PMAPI PM_getMachineName(void)
```

### Prototype In

pmapi.h

### Return Value

Constant string pointer to the network machine name

### Description

This function is used to obtain the machine name for the computer on the network if possible. This is not always possible for all OS'es (especially when the OS has no networking!), so in some cases this will simply be a constant value if the network machine name cannot be determined.

## PM\_getOSName

Return the name of the operating system environment.

### Declaration

```
char * PMAPI PM_getOSName(void)
```

### Prototype In

pmapi.h

### Return Value

String representing the operating system name

### Description

This function returns a string representation of the name of the runtime operating system environment. This is useful for binary portable code that needs to display or log the operating system name for informational purposes.

### See Also

*PM\_getOSType*

## PM\_getOSType

Return the operating system type identifier.

### Declaration

```
long PMAPI PM_getOSType(void)
```

### Prototype In

pmapi.h

### Return Value

Flag representing the OS type

### Description

This function returns a flag that represents the operating system type, so that binary portable code that does need to handle OS dependencies internally can do so with runtime checks if necessary. Please see the %SCITECH%\include\drvlib\os\os.h header file for the current definition of operating system types supported. This list will grow as more operating systems are supported by the PM library.

### See Also

*PM\_getOSName*

## PM\_getPhysicalAddr

Find the physical address of a linear memory address for the current process.

### Declaration

```
ulong PMAPI PM_getPhysicalAddr(  
    void *p)
```

### Prototype In

pmapi.h

### Parameters

*p*                      Linear address to convert

### Return Value

Physical memory address, or PM\_BAD\_PHYS\_ADDRESS on error.

### Description

This function is used to convert a linear address pointer to a physical memory address. If this fails, or is not supported for some reason, this function will return a value of PM\_BAD\_PHYS\_ADDRESS.

### See Also

*PM\_mapPhysicalAddr, PM\_getPhysicalAddrRange*

## PM\_getPhysicalAddrRange

Find physical addresss of a linear memory address for the current process.

### Declaration

```
ibool PMAPI PM_getPhysicalAddrRange (
    void *p,
    ulong length,
    ulong *physAddress)
```

### Prototype In

pmapi.h

### Parameters

<i>p</i>	Linear address to convert
<i>length</i>	Length of memory region to convert
<i>physAddress</i>	Array to store physical addresses into

### Return Value

True on success, false on error.

### Description

This function is used to convert a large linear address pointer to a list of physical memory addresses. The list of addresses will be one per page for the linear address, and the addresses will all be page aligned. This is useful to convert a single linear address block into the list of physical memory pages for the memory to be programmed into DMA operations etc.

### See Also

*PM\_mapPhysicalAddr*, *PM\_getPhysicalAddr*

## PM\_getProcAddress

Get the address of a named procedure from a shared library.

### Declaration

```
void * PMAPI PM_getProcAddress(  
    PM_MODULE hModule,  
    const char *szProcName)
```

### Prototype In

pmapi.h

### Parameters

<i>hModule</i>	Handle to the module to get procedure from
<i>szProcName</i>	Name of the procedure to get address of

### Return Value

Pointer to the start of the function in the shared library

### Description

This function is used to get the address of a named function in a shared library that was loaded with the *PM\_loadLibrary*.

### See Also

*PM\_loadLibrary*, *PM\_freeLibrary*

## PM\_getSNAPConfigPath

Return the path to the SNAP configuration files.

### Declaration

```
const char * PMAPI PM_getSNAPConfigPath(void)
```

### Prototype In

pmapi.h

### Return Value

Constant string pointer to the SNAP configuration files path

### Description

This function is used to obtain the standard path where the SNAP configuration files should be found. This is usually operating system specific, but it can be overridden for debugging and development purposes using the SNAP\_PATH environment variable. In most cases this is a 'config' directory below the directory reported by *PM\_getSNAPPath*, however on some network operating systems with shared directories, the config directories may be specific to each user while the SNAP binaries live in a shared directory common to all users. This is the case for instance under the QNX operating systems.

### See Also

*PM\_getSNAPPath*

## PM\_getSNAPPath

Return the path to the SNAP driver files.

### Declaration

```
const char * PMAPI PM_getSNAPPath(void)
```

### Prototype In

pmapi.h

### Return Value

Constant string pointer to the SNAP path

### Description

This function is used to obtain the standard path where the SNAP drivers should be found. This is usually operating system specific, but it can be overridden for debugging and development purposes using the SNAP\_PATH environment variable.

### See Also

*PM\_getSNAPConfigPath*



## PM\_getUniqueID

Return a unique identifier for the machine if possible.

### Declaration

```
const char * PMAPI PM_getUniqueID(void)
```

### Prototype In

pmapi.h

### Return Value

Constant string pointer to the unique identifier

### Description

This function is used to obtain a unique identifier string for the computer on the network if possible. This is not always possible for all OS'es (especially when the OS has no networking!), so in some cases this will simply be a constant value if the network machine name or unique ID cannot be determined.

## PM\_getVESABuf

Allocate the real mode VESA transfer buffer for communicating with the BIOS.

### Declaration

```
void * PMAPI PM_getVESABuf(  
    uint *len,  
    uint *rseg,  
    uint *roff)
```

### Prototype In

pmapi.h

### Parameters

<i>len</i>	Place to store the length of the VESA buffer
<i>rseg</i>	Place to store the real mode segment of the VESA buffer
<i>roff</i>	Place to store the real mode offset of the VESA buffer

### Return Value

Pointer to the transfer buffer on success, NULL on failure.

### Description

This function is used to allocate the real mode VESA transfer buffer for communicating with the underlying real mode Video BIOS. If the operating system cannot support accessing the VESA BIOS functions, this function will return NULL. If this function does succeed, the length of the buffer will be returned in the len parameter while the real mode segment and offset of the buffer will be returned in the rseg and roff parameters. A regular C pointer to the buffer is returned directly and can be used to read and write data from the transfer buffer.

## PM\_getVGASize

Get the size of the VGA hardware state save buffer

### Declaration

```
int PMAPI PM_getVGASize(void)
```

### Prototype In

pmapi.h

### Description

Returns the size of the VGA state buffer.

### See Also

*PM\_saveVGASize, PM\_restoreVGASize*

## PM\_getch

Wait for and return the next keypress.

### Declaration

```
int PMAPI PM_getch(void)
```

### Prototype In

pmapi.h

### Return Value

ASCII code for the key that was pressed

### Description

This function waits for and returns the next keypress, and returns the ASCII code of the key that was pressed. This function is valid only for operating systems that support running in console modes (DOS, Linux, OS/2 etc).

## PM\_getdcwd

Function to get the current working directory for the specified drive.

### Declaration

```
void PMAPI PM_getdcwd(  
    int drive,  
    char *dir,  
    int len)
```

### Prototype In

pmapi.h

### Parameters

<i>drive</i>	Drive letter to get working directory for
<i>dir</i>	Place to store working directory
<i>len</i>	Length of working directory buffer

### Description

This function is used to get the current working directory for the specified drive from the operating system. Under Unix systems, this will always return the current working directory regardless of what the value of 'drive' is since there is no concept of drives under Unix.

### See Also

*PM\_getCurrentPath*

## PM\_haveBIOSAccess

Determines if access to the real mode BIOS is available

### Declaration

```
ibool PMAPI PM_haveBIOSAccess(void)
```

### Prototype In

pmapi.h

### Return Value

True if the system provides BIOS access, false if not.

### Description

This function is used to determine if the operating system can provide access to the real mode BIOS or not. Many operating systems can provide full access (DOS, Windows 9x, Linux), others can provide limited access (OS/2) while other still provide no access (Windows NT/2000/XP, QNX etc).

If you need to call the *PM\_int86* functions, you can first call this function to determine if the BIOS access is available or not. If this function returns false, do **not** call the *PM\_int86* functions!

## PM\_init

Initialise the PM library

### Declaration

```
void PMAPI PM_init(void)
```

### Prototype In

pmapi.h

### Description

Initialise the PM library and connect to our helper device driver. If we cannot connect to our helper device driver, we bail out with an error message. Our Windows 9x VxD is dynamically loadable, so it can be loaded after the system has started. On Windows NT/2000/XP the device driver service must be installed first during application installation by a system administrator. On OS/2 the SDDHELP.SYS driver must be installed in the CONFIG.SYS file prior to use. On other platforms this function does not usually require an external device driver and just initialises the PM library internals.

## PM\_inpb

Read a byte value from an I/O port

### Declaration

```
uchar PMAPI PM_inpb(  
    int port)
```

### Prototype In

pmapi.h

### Parameters

*port*                    I/O port to read the value from

### Return Value

Byte value read from the I/O port

### Description

This function is used to read a byte value from an I/O port.

### See Also

*PM\_inpw, PM\_inpd, PM\_outpb*



## PM\_inpd

Read a double word value from an I/O port

### Declaration

```
ulong PMAPI PM_inpd(  
    int port)
```

### Prototype In

pmapi.h

### Parameters

*port*                      I/O port to read the value from

### Return Value

Double word value read from the I/O port

### Description

This function is used to read a double word value from an I/O port.

### See Also

*PM\_inpb, PM\_inpw, PM\_outpb*

## PM\_inpw

Read a word value from an I/O port

### Declaration

```
ushort PMAPI PM_inpw(  
    int port)
```

### Prototype In

pmapi.h

### Parameters

*port*                    I/O port to read the value from

### Return Value

Word value read from the I/O port

### Description

This function is used to read a word value from an I/O port.

### See Also

*PM\_inpb, PM\_inpd, PM\_outpb*

## PM\_installService

Installs a Windows NT/2000/XP service.

### Declaration

```
ulong PMAPI PM_installService(
    const char *szDriverName,
    const char *szServiceName,
    const char *szLoadGroup,
    ulong dwServiceType)
```

### Prototype In

pmapi.h

### Parameters

<i>szDriverName</i>	Actual name of the driver to install in the system
<i>szServiceName</i>	Name of the service to create
<i>szLoadGroup</i>	Load group for the driver (NULL for normal drivers)
<i>dwServiceType</i>	Service type to create

### Return Value

ERROR\_SUCCESS on success, error code on failure.

### Description

This function does all the work to install the system service into the system (ie: a Windows NT style device driver). The driver is not however activated; for that you must use the *PM\_startService* function. This version always creates the service with the SERVICE\_BOOT\_START start type.

---

**Note:** *This function is Windows specific! It is quite useful so it is documented here.*

---

### See Also

*PM\_installServiceExt, PM\_startService, PM\_stopService, PM\_removeService*

## PM\_installServiceExt

Installs a Windows NT/2000/XP service.

### Declaration

```
ulong PMAPI PM_installServiceExt(
    const char *szDriverName,
    const char *szServiceName,
    const char *szLoadGroup,
    ulong dwServiceType,
    ulong dwStartType)
```

### Prototype In

pmapi.h

### Parameters

<i>szDriverName</i>	Actual name of the driver to install in the system
<i>szServiceName</i>	Name of the service to create
<i>szLoadGroup</i>	Load group for the driver (NULL for normal drivers)
<i>dwServiceType</i>	Service type to create
<i>dwStartType</i>	Service start type to create

### Return Value

ERROR\_SUCCESS on success, error code on failure.

### Description

This function does all the work to install the system service into the system (ie: a Windows NT style device driver). The driver is not however activated; for that you must use the *PM\_startService* function. This version also allows you to specify the service start type.

---

**Note:** *This function is Windows specific! It is quite useful so it is documented here.*

---

### See Also

*PM\_installService, PM\_startService, PM\_stopService, PM\_removeService*

## PM\_int86

Execute a real mode software interrupt.

### Declaration

```
int PMAPI PM_int86(
    int intno,
    RMREGS *in,
    RMREGS *out)
```

### Prototype In

pmapi.h

### Parameters

<i>intno</i>	Software interrupt number to execute
<i>in</i>	Register block to load before calling interrupt
<i>out</i>	Register block to load with registers after interrupt was called

### Return Value

Value returned in the EAX register.

### Description

This function is used to execute a real mode software interrupt, which is used to call the real mode BIOS functions and drivers directly. If you make calls to the real mode BIOS functions or drivers with this function, there is *no* parameter translation at all. Hence you need to translate any real mode memory pointers etc passed into and returned from this function with the *PM\_allocRealSeg* and *PM\_mapRealPointer* functions.

When this function executes the real mode software interrupt, the machine registers will be loaded with the values passed in the 'in' parameter. When the interrupt completes, the values in the machine registers will then be saved into the 'out' parameter.

This is only supported for operating systems that support BIOS access (ie: the *PM\_haveBIOSAccess* function returns true).

### See Also

*PM\_int86x*, *PM\_callRealMode*, *PM\_allocRealSeg*

## PM\_int86x

Execute a real mode software interrupt with segment registers

### Declaration

```
int PMAPI PM_int86x(
    int intno,
    RMREGS *in,
    RMREGS *out,
    RMSREGS *sregs)
```

### Prototype In

pmapi.h

### Parameters

<i>intno</i>	Software interrupt number to execute
<i>in</i>	Register block to load before calling interrupt
<i>out</i>	Register block to load with registers after interrupt was called
<i>sregs</i>	Segment register block to load and return values in

### Return Value

Value returned in the EAX register.

### Description

This function is used to execute a real mode software interrupt, which is used to call the real mode BIOS functions and drivers directly. If you make calls to the real mode BIOS functions or drivers with this function, there is *no* parameter translation at all. Hence you need to translate any real mode memory pointers etc passed into and returned from this function with the *PM\_allocRealSeg* and *PM\_mapRealPointer* functions.

When this function executes the real mode software interrupt, the machine registers will be loaded with the values passed in the 'in' parameter. When the interrupt completes, the values in the machine registers will then be saved into the 'out' parameter.

This version also allows you to pass down segment register values to be passed to the real mode code with the *sregs* parameter. On return from this function, *sregs* will contain the segment registers that were returned from the function when the real mode interrupt was completed.

This is only supported for operating systems that support BIOS access (ie: the *PM\_haveBIOSAccess* function returns true).

### See Also

*PM\_int86*, *PM\_callRealMode*, *PM\_allocRealSeg*

## PM\_isSDDActive

Returns true if SNAP Graphics is the active display driver in the system.

### Declaration

```
ibool PMAPI PM_isSDDActive(void)
```

### Prototype In

pmapi.h

### Return Value

True if SNAP Graphics is active, false if not.

### Description

This function is used to determine if the SNAP Graphics display drivers are the active display drivers in the system or not.

## PM\_kbhit

Check if a key has been pressed.

### Declaration

```
int PMAPI PM_kbhit(void)
```

### Prototype In

pmapi.h

### Return Value

True if a key was pressed, false if not.

### Description

This function check if a key has been pressed. This function is valid only for operating systems that support running in console modes (DOS, Linux, OS/2 etc).



## PM\_loadDirectDraw

Loads the DirectDraw libraries

### Declaration

```
void * PMAPI PM_loadDirectDraw(  
    int device)
```

### Prototype In

pmapi.h

### Parameters

*device*                      Index of the device to load DirectDraw for (0 for primary)

### Return Value

Pointer to the loaded DirectDraw library

### Description

Attempts to dynamically load the DirectDraw DLL's and create the DirectDraw objects that we need. This function is generally never called by application code, but is called by the DirectX SNAP drivers when the DirectX libraries need to be loaded.

---

**Note:** *This function is Windows specific*

---

### See Also

*PM\_unloadDirectDraw, PM\_getDirectDrawWindow*

## PM\_loadLibrary

Load an OS specific shared library or DLL.

### Declaration

```
PM_MODULE PMAPI PM_loadLibrary(  
    const char *szDLLName)
```

### Prototype In

pmapi.h

### Parameters

*szDLLName*                      Name of the OS specific library to load

### Return Value

Pointer to the loaded module handle, NULL on failure.

### Description

This function is used to load an operating system specific shared library or DLL. This is mostly used by binary portable code that needs to directly interface to operating system specific shared library code.

If the OS does not support shared libraries, this function simply returns NULL.

### See Also

*PM\_getProcAddress, PM\_freeLibrary*

## PM\_lockCodePages

Lock code pages so they won't be paged to disk

### Declaration

```
int PMAPI PM_lockCodePages(  
    __codePtr p,  
    uint len,  
    PM_lockHandle *lh)
```

### Prototype In

pmapi.h

### Parameters

<i>p</i>	Linear pointer to the memory to lock down
<i>len</i>	Length of the memory block to lock down
<i>lh</i>	Pointer to the lock handle returned

### Description

This function is used to lock a block of memory such that it will not be paged to disk by the operating systems virtual memory manager. This is mostly used such that interrupt handlers in device drivers and the data used by the interrupt handlers will never be paged out to disk.

This version is used to lock code pages in memory.

### See Also

*PM\_unlockCodePages*, *PM\_lockDataPages*

## PM\_lockDataPages

Lock data pages so they won't be paged to disk

### Declaration

```
int PMAPI PM_lockDataPages(  
    void *p,  
    uint len,  
    PM_lockHandle *lh)
```

### Prototype In

pmapi.h

### Parameters

<i>p</i>	Linear pointer to the memory to lock down
<i>len</i>	Length of the memory block to lock down
<i>lh</i>	Pointer to the lock handle returned

### Description

This function is used to lock a block of memory such that it will not be paged to disk by the operating systems virtual memory manager. This is mostly used such that interrupt handlers in device drivers and the data used by the interrupt handlers will never be paged out to disk.

This version is used to lock data pages in memory.

### See Also

*PM\_unlockDataPages*, *PM\_lockCodePages*

## PM\_makepath

Make a full pathname from split components.

### Declaration

```
void PMAPI PM_makepath(
    char *path,
    const char *drive,
    const char *dir,
    const char *name,
    const char *ext)
```

### Prototype In

pmapi.h

### Parameters

<i>path</i>	Place to store full path
<i>drive</i>	Drive component for path
<i>dir</i>	Directory component for path
<i>name</i>	Filename component for path
<i>ext</i>	Extension component for path

### Description

Function to make a full pathname from split components. Under Unix the drive component will usually be empty. If the drive, dir, name, or ext parameters are null or empty, they are not inserted in the path string. Otherwise, if the drive doesn't end with a colon, one is inserted in the path. If the dir doesn't end in a slash, one is inserted in the path. If the ext doesn't start with a dot, one is inserted in the path.

The maximum sizes for the path string is given by the constant PM\_MAX\_PATH, which includes space for the null-terminator.

### See Also

PM\_splitPath

## PM\_malloc

Allocate a block of memory.

### Declaration

```
void * PMAPI PM_malloc(  
    size_t size)
```

### Prototype In

pmapi.h

### Parameters

*size*                      Size of block to allocate in bytes

### Return Value

Pointer to allocated block, or NULL if out of memory.

### Description

Allocates a block of memory of length *size*. If you have changed the memory allocation routines with the *PM\_useLocalMalloc* function, then calls to this function will actually make calls to the local memory allocation routines that you have registered.

### See Also

*PM\_calloc*, *PM\_realloc*, *PM\_free*, *PM\_useLocalMalloc*

## PM\_mallocShared

Allocate a block of system global shared memory

### Declaration

```
void * PMAPI PM_mallocShared(  
    long size)
```

### Prototype In

pmapi.h

### Parameters

*size*                      Size of the shared memory block to allocate

### Return Value

Pointer to the shared memory block, NULL on failure.

### Description

This function is used to allocate a block of shared memory, such that the linear address returned for this shared memory is *identical* for all processes in the system. If this cannot be provided, this function will return NULL.

### See Also

*PM\_freeShared*

## PM\_mapPhysicalAddr

Map a physical address to a linear address in the callers process.

### Declaration

```
void * PMAPI PM_mapPhysicalAddr(
    ulong base,
    ulong limit,
    ibool isCached)
```

### Prototype In

pmapi.h

### Parameters

<i>base</i>	Physical base address of the memory to map
<i>limit</i>	Limit for the mapped memory region (length-1)
<i>isCached</i>	True if the memory should be cached, false if not

### Return Value

Pointer to the mapped memory, false on failure.

### Description

This function is used to obtain a pointer to the any physical memory location in the computer, mapped into the linear address space of the calling process. If the *isCached* parameter is set to true, caching will be enabled for this region. If this parameter is off, caching will be disabled. Caching must always be disabled when accessing memory mapped registers, as they cannot be cached. Note that this does not enable write combing for the region; for that you need to call the *PM\_enableWriteCombine* function (however caching must be enabled before the write combining will work!).

### See Also

*PM\_freePhysicalAddr*, *PM\_getPhysicalAddr*



## PM\_mapRealPointer

Map a real mode pointer to a protected mode pointer.

### Declaration

```
void * PMAPI PM_mapRealPointer(  
    uint r_seg,  
    uint r_off)
```

### Prototype In

pmapi.h

### Parameters

<i>r_seg</i>	Real mode segment address to map
<i>r_off</i>	Real mode segment offset to map

### Description

This function is used to map a real mode pointer in segment:offset format into a protected mode linear address. This is only supported for operating systems that support BIOS access (ie: the *PM\_haveBIOSAccess* function returns true).

### See Also

*PM\_allocRealSeg*

## PM\_mkdir

Function to create a directory.

### Declaration

```
ibool PMAPI PM_mkdir(  
    const char *filename)
```

### Prototype In

pmapi.h

### Parameters

*filename*                      Full path to filename for directory to create

### Return Value

True on success, false on failure.

### Description

This function is used to create a new directory in the file system.

### See Also

*PM\_rmdir*

## PM\_openConsole

Opens a console for windowed or fullscreen operation

### Declaration

```
PM_HWND WINAPI PM_openConsole(
    PM_HWND hWndUser,
    int device,
    int xRes,
    int yRes,
    int bpp,
    ibool fullScreen)
```

### Prototype In

pmapi.h

### Parameters

<i>hWndUser</i>	Pointer to use application window (NULL if none)
<i>device</i>	Index of the device to control (0 for primary)
<i>xRes</i>	X resolution planned for the fullscreen console mode
<i>yRes</i>	Y resolution planned for the fullscreen console mode
<i>bpp</i>	Color depth planned for the fullscreen console mode
<i>fullScreen</i>	True if the console is fullscreen, false if windowed

### Return Value

Pointer to the console window handle

### Description

This function open a console for output to the screen, creating the main event handling window if necessary when the hWndUser parameter is set to NULL.

### See Also

*PM\_getConsoleStateSize*, *PM\_saveConsoleState*, *PM\_setSuspendAppCallback*,  
*PM\_restoreConsoleState*, *PM\_closeConsole*

## PM\_outpb

Write a byte value to an I/O port

### Declaration

```
void PMAPI PM_outpb(  
    int port,  
    uchar val)
```

### Prototype In

pmapi.h

### Parameters

<i>port</i>	I/O port to read the value from
<i>val</i>	Value to write to the I/O port

### Description

This function is used to write a byte value to an I/O port.

### See Also

*PM\_inpb, PM\_outpw, PM\_outpd*

## PM\_outpd

Write a double word value to an I/O port

### Declaration

```
void PMAPI PM_outpd(  
    int port,  
    ulong val)
```

### Prototype In

pmapi.h

### Parameters

<i>port</i>	I/O port to read the value from
<i>val</i>	Value to write to the I/O port

### Description

This function is used to write a double word value to an I/O port.

### See Also

*PM\_inpb, PM\_outpb, PM\_outpw*

## PM\_outpw

Write a word value to an I/O port

### Declaration

```
void PMAPI PM_outpw(  
    int port,  
    ushort val)
```

### Prototype In

pmapi.h

### Parameters

<i>port</i>	I/O port to read the value from
<i>val</i>	Value to write to the I/O port

### Description

This function is used to write a word value to an I/O port.

### See Also

*PM\_inpb, PM\_outpb, PM\_outpd*

## PM\_realloc

Re-allocate a block of memory

### Declaration

```
void * PMAPI PM_realloc(  
    void *ptr,  
    size_t size)
```

### Prototype In

pmapi.h

### Parameters

<i>ptr</i>	Pointer to block to resize
<i>size</i>	size of unit in bytes

### Return Value

Pointer to allocated memory if successful, NULL if out of memory.

### Description

This function reallocates a block of memory that has been previously been allocated to the new of size. The new size may be smaller or larger than the original block of memory. If you have changed the memory allocation routines with the *PM\_useLocalMalloc* function, then calls to this function will actually make calls to the local memory allocation routines that you have registered.

### See Also

*PM\_malloc, PM\_calloc, PM\_free, PM\_useLocalMalloc*

## PM\_removeService

Removes a Windows NT/2000/XP service.

### Declaration

```
ulong PMAPI PM_removeService(  
    const char *szServiceName)
```

### Prototype In

pmapi.h

### Parameters

*szServiceName*                      Name of the service to start

### Return Value

ERROR\_SUCCESS on success, error code on failure.

### Description

This function is used to remove a service completely from the system.

---

**Note:** *This function is Windows specific! It is quite useful so it is documented here.*

---

### See Also

*PM\_installServiceExt, PM\_startService, PM\_stopService*



## PM\_restartRealTimeClock

Restarts the real time clock ticking again

### Declaration

```
void PMAPI PM_restartRealTimeClock(  
    int frequency)
```

### Prototype In

pmapi.h

### Description

This function is used to restart the real time clock ticking. Note that when we are actually using IRQ0 instead, this functions does nothing.

### See Also

*PM\_setRealTimeClockHandler, PM\_setRealTimeClockFrequency, PM\_stopRealTimeClock, PM\_restoreRealTimeClockHandler*

## PM\_restoreConsoleState

Restores the state of the OS console.

### Declaration

```
void PMAPI PM_restoreConsoleState(  
    const void *stateBuf,  
    PM_HWND hwndConsole)
```

### Prototype In

pmapi.h

### Parameters

<i>stateBuf</i>	State buffer to restore state from
<i>hwndConsole</i>	Console window handle

### Description

This function restore the state of the OS console that was previously saved with the *PM\_saveConsoleState* function.

### See Also

*PM\_openConsole*, *PM\_getConsoleStateSize*, *PM\_setSuspendAppCallback*,  
*PM\_saveConsoleState*, *PM\_closeConsole*

## **PM\_restoreRealTimeClockHandler**

Restore the original real time clock handler.

### **Declaration**

```
void PMAPI PM_restoreRealTimeClockHandler(void)
```

### **Prototype In**

pmapi.h

### **Description**

This function is used to restore the original real time clock handler.

### **See Also**

*PM\_setRealTimeClockHandler*

## PM\_restoreThreadPriority

Restore the original thread priority.

### Declaration

```
void PMAPI PM_restoreThreadPriority(  
    ulong oldPriority)
```

### Prototype In

pmapi.h

### Parameters

*oldPriority*                      Old thread priority to restore

### Description

This function is used to restore the current thread priority to the previous value.

### See Also

PM\_setThreadPriority

## PM\_restoreVGASState

Restore the VGA hardware state from the save buffer

### Declaration

```
void PMAPI PM_restoreVGASState(  
    const void *stateBuf)
```

### Prototype In

pmapi.h

### Parameters

*stateBuf*                      Save buffer to restore the state of the VGA hardware from

### Description

Restores the state of all VGA compatible registers from the save buffer passed in the 'stateBuf' parameter.

### See Also

*PM\_getVGASStateSize*, *PM\_saveVGASState*

## PM\_rmdir

Function to remove a directory.

### Declaration

```
ibool PMAPI PM_rmdir(  
    const char *filename)
```

### Prototype In

pmapi.h

### Parameters

*filename*                      Full path to filename for directory to remove

### Return Value

True on success, false on failure.

### Description

This function is used to remove a directory from the file system. This function will fail unless the directory is empty.

### See Also

*PM\_mkdir*

## PM\_runningInAWindow

Determines if the application is running in a window.

### Declaration

```
ibool PMAPI PM_runningInAWindow(void)
```

### Prototype In

pmapi.h

### Return Value

True if running in a window, false if not.

### Description

This function is primarily used for console programs that need to know if they are running in a fullscreen console mode or in a window under a GUI environment. Presently this function is implemented for DOS and OS/2 console mode programs. It could also be implemented for Linux console apps also.

## PM\_saveConsoleState

Save the state of the OS console.

### Declaration

```
void PMAPI PM_saveConsoleState(  
    void *stateBuf,  
    PM_HWND hwndConsole)
```

### Prototype In

pmapi.h

### Parameters

<i>stateBuf</i>	State buffer to save state to
<i>hwndConsole</i>	Console window handle

### Description

This function saves the state of the OS console, so that it can be later restored by the *PM\_restoreConsoleState*. This function must be called to save the console state so that it can go into graphics mode. On many OS'es this doesn't do much, but on Linux for instance this properly enabled the console for graphics output, and allows us to properly restore it later.

### See Also

*PM\_openConsole*, *PM\_getConsoleStateSize*, *PM\_setSuspendAppCallback*,  
*PM\_restoreConsoleState*, *PM\_closeConsole*



## PM\_saveVGASState

Save the VGA hardware state into a save buffer

### Declaration

```
void PMAPI PM_saveVGASState(  
    void *stateBuf)
```

### Prototype In

pmapi.h

### Parameters

*stateBuf*                      Place to save the state of the VGA hardware

### Description

Save the state of all VGA compatible registers into save buffer passed in the 'stateBuf' parameter. You must first call the *PM\_getVGASStateSize* function to allocate a buffer big enough to hold the VGA hardware state before you call this function.

### See Also

*PM\_getVGASStateSize*, *PM\_restoreVGASState*

## PM\_setDebugLog

Sets the location of the debug log file.

### Declaration

```
void PMAPI PM_setDebugLog(  
    const char *logFilePath)
```

### Prototype In

pmapi.h

### Parameters

*logFilePath*                      Full file and path name to debug log file.

### Description

Sets the name and location of the debug log file. The debug log file is created and written to when runtime checks, warnings and failure conditions are logged to disk when code is compiled in CHECKED mode. By default the log file is called 'scitech.log' and goes into the current SciTech SNAP path for the application. You can use this function to set the filename and location of the debug log file to your own application specific directory.

## PM\_setFatalErrorCleanup

Add a user defined *PM\_fatalError* cleanup function.

### Declaration

```
void PMAPI PM_setFatalErrorCleanup(  
    void (PMAPI cleanup)(void));  
void PMAPI PM_setFatalErrorCleanup(PM_fatalCleanupHandler cleanup)
```

### Prototype In

pmapi.h

### Parameters

*cleanup*                      New fatal error cleanup function to use

### Description

This function is provided to allow a user defined fatal error cleanup function to be registered. If any code call *PM\_fatalError*, this cleanup function will be called first, allowing the cleanup function to put the operating system back into a valid state before displaying the fatal error message.

### See Also

*PM\_fatalError*

## PM\_setFileAttr

Function to change the file attributes for a specific file.

### Declaration

```
void PMAPI PM_setFileAttr(  
    const char *filename,  
    uint attrib)
```

### Prototype In

pmapi.h

### Parameters

<i>filename</i>	Full path to filename for file to change
<i>attrib</i>	New attributes for the file ( <i>PMFileFlagsType</i> )

### Description

This function is used to file attributes for a specific file to the values passed in the attribute parameter (a combination of flags defined in *PMFileFlagsType*). Under Unix system some of these flags are ignored, such as the hidden and system attributes.

### See Also

*PM\_getFileAttr*

## PM\_setFileTime

Function to set the file time and date for a specific file.

### Declaration

```
ibool PMAPI PM_setFileTime(  
    const char *filename,  
    ibool gmTime,  
    PM_time *time)
```

### Prototype In

pmapi.h

### Parameters

<i>filename</i>	Full path to filename for file to set date and time for
<i>gmTime</i>	True if time should be in the GMT timezone
<i>time</i>	Time to set for the file

### Return Value

True on success, false on failure.

### Description

This function is used to set the file date and time stamp for a specific file. If the gmTime parameter is true, the time passed in should be in the GMT time zone, otherwise it is in the local machine time zone.

### See Also

*PM\_getFileTime*

## PM\_setIOPL

Set the I/O privilege level for the current process

### Declaration

```
int PMAPI PM_setIOPL(  
    int iopl)
```

### Prototype In

pmapi.h

### Parameters

*iopl*                      New IOPL to make active (0 - 3)

### Return Value

Previous IOPL active before the change was made

### Description

This function is used to change the I/O privilege level of the current process, so that it can access I/O ports directly. This works on all supported OS'es to date, even on OS/2 and Windows NT/2000/XP, provided you have the necessary kernel level drivers or services installed.

### See Also

*PM\_getIOPL*

## PM\_setLocalBPDPATH

Function to override the SNAP BPD driver path

### Declaration

```
void PMAPI PM_setLocalBPDPATH(  
    const char *path)
```

### Prototype In

pmapi.h

### Parameters

*path*                      Local path to the SciTech SNAP BPD driver files.

### Description

This function is used by the application program to override the location of the SciTech SNAP driver files that are loaded. Normally the loader code will look in the system SciTech SNAP directories first, then in the 'drivers' directory relative to the current working directory, and finally relative to the MGL\_ROOT environment variable. By default the local BPD path is always set to the current directory if not initialised.

### See Also

*PM\_findBPD*

## **PM\_setMaxThreadPriority**

Increase the thread priority to maximum, if possible.

### **Declaration**

```
ulong PMAPI PM_setMaxThreadPriority(void)
```

### **Prototype In**

pmapi.h

### **Return Value**

Old thread priority

### **Description**

This function is used to set the current thread priority to the maximum possible. This should not be used very often, but is useful for important timing and calibration loops that need to be very accurate. The current thread priority that was active before the change is returned.

### **See Also**

*PM\_restoreThreadPriority*



## PM\_setOSCursorLocation

Set the location of the OS text mode console cursor.

### Declaration

```
void PM_setOSCursorLocation(  
    int x,  
    int y);  
void PMAPI PM_setOSCursorLocation(int x,int y)
```

### Prototype In

pmapi.h

### Parameters

<i>x</i>	New console cursor X coordinate
<i>y</i>	New console cursor Y coordinate

### Description

This function is used to set the location of the OS text mode console cursor. This function is valid only for operating systems that support running in console modes (DOS, Linux, OS/2 etc).

## PM\_setOSScreenWidth

Set the dimensions of the OS text mode console.

### Declaration

```
void PM_setOSScreenWidth(  
    int width,  
    int height);  
void PMAPI PM_setOSScreenWidth(int width,int height)
```

### Prototype In

pmapi.h

### Parameters

<i>width</i>	New width of the OS text mode console
<i>height</i>	New height of the OS text mode console

### Description

This function set the width and height of the OS text mode console. This should be done if some method other than OS provided functions is used to change the console mode (ie: 80x50 or 80x60 instead of 80x25), so that the operating system functions themselves will know how to output to the new mode. This function is valid only for operating systems that support running in console modes (DOS, Linux, OS/2 etc).

## PM\_setRealTimeClockFrequency

Set the real time clock frequency (for stereo modes).

### Declaration

```
void PMAPI PM_setRealTimeClockFrequency(  
    int frequency)
```

### Prototype In

pmapi.h

### Parameters

*frequency*                      New frequency to program the RTC to run at

### Description

This function is used to change the real time clock frequency that is used for software stereo modes by the SNAP Graphics drivers. The interrupt handler must first be installed with the *PM\_setRealTimeClockHandler* function.

### See Also

*PM\_setRealTimeClockHandler*, *PM\_stopRealTimeClock*, *PM\_restartRealTimeClock*,  
*PM\_restoreRealTimeClockHandler*

## PM\_setRealTimeClockHandler

Set the real time clock handler (used for software stereo modes).

### Declaration

```
ibool PMAPI PM_setRealTimeClockHandler(  
    PM_intHandler ih,  
    int frequency)
```

### Prototype In

pmapi.h

### Parameters

<i>ih</i>	New C based interrupt handler to install
<i>frequency</i>	New frequency to program the RTC to run at

### Return Value

True on success, false on failure.

### Description

This function is used to set the real time clock handler that is used for software stereo modes by the SNAP Graphics drivers. This is presently only supported under DOS, Windows 9x and Windows NT/2000/XP environments. It is also not supported by general application programs, only by device driver environments (ie: Win32 apps cannot use this function, only Windows 9x VxD drivers or Windows NT style kernel drivers).

### See Also

*PM\_setRealTimeClockFrequency*, *PM\_stopRealTimeClock*, *PM\_restartRealTimeClock*, *PM\_restoreRealTimeClockHandler*

## PM\_setSuspendAppCallback

Set the suspend application callback for the fullscreen console.

### Declaration

```
void PMAPI PM_setSuspendAppCallback(  
    PM_suspendApp_cb saveState)
```

### Prototype In

pmapi.h

### Description

This function set the suspend application callback for the fullscreen console. This callback is used to allow the application to properly save and restore it's own state when the fullscreen console is being switch away from or being switched back to.

### See Also

*PM\_openConsole, PM\_closeConsole*

## PM\_sleep

Sleep for the specified number of milliseconds.

### Declaration

```
void PMAPI PM_sleep(  
    ulong milliseconds)
```

### Prototype In

pmapi.h

### Parameters

<i>milliseconds</i>	Number of milliseconds to sleep for
---------------------	-------------------------------------

### Description

This function is used to pause the current process and put it to sleep for the specified number of milliseconds.

## PM\_splitpath

Split a full pathname into components.

### Declaration

```
int PMAPI PM_splitpath(
    const char *path,
    char *drive,
    char *dir,
    char *name,
    char *ext)
```

### Prototype In

pmapi.h

### Parameters

<i>path</i>	Full path to split
<i>drive</i>	Drive component for path
<i>dir</i>	Directory component for path
<i>name</i>	Filename component for path
<i>ext</i>	Extension component for path

### Return Value

Flags indicating what components were parsed (*PMSplitPathFlags*)

### Description

Function to split a full pathname into separate components in the form

`X:\DIR\SUBDIR\NAME.EXT`

and splits path into its four components. It then stores those components in the strings pointed to by drive, dir, name and ext. (Each component is required but can be a NULL, which means the corresponding component will be parsed but not stored).

The maximum sizes for these strings are given by the constants PM\_MAX\_DRIVE and PM\_MAX\_PATH. PM\_MAX\_DRIVE is always 4, and PM\_MAX\_PATH is usually at least 256 characters. Under Unix the dir, name and ext components may be up to the full path in length.

### See Also

PM\_makePath

## PM\_startService

Starts a Windows NT/2000/XP service.

### Declaration

```
ulong PMAPI PM_startService(  
    const char *szServiceName)
```

### Prototype In

pmapi.h

### Parameters

*szServiceName*                      Name of the service to start

### Return Value

ERROR\_SUCCESS on success, error code on failure.

### Description

This function is used to start the specified service and make it active.

---

**Note:** *This function is Windows specific! It is quite useful so it is documented here.*

---

### See Also

*PM\_installServiceExt, PM\_stopService, PM\_removeService*



## PM\_stopRealTimeClock

Stops the real time clock from ticking

### Declaration

```
void PMAPI PM_stopRealTimeClock(void)
```

### Prototype In

pmapi.h

### Description

This function is used to stops the real time clock from ticking. Note that when we are actually using IRQ0 instead, this functions does nothing (unlike calling *PM\_setRealTimeClockFrequency* directly).

### See Also

*PM\_setRealTimeClockHandler*, *PM\_setRealTimeClockFrequency*, *PM\_restartRealTimeClock*, *PM\_restoreRealTimeClockHandler*

## PM\_stopService

Stops a Windows NT/2000/XP service.

### Declaration

```
ulong PMAPI PM_stopService(  
    const char *szServiceName)
```

### Prototype In

pmapi.h

### Parameters

*szServiceName*                      Name of the service to start

### Return Value

ERROR\_SUCCESS on success, error code on failure.

### Description

This function is used to stop the specified service and disable it.

---

**Note:** *This function is Windows specific! It is quite useful so it is documented here.*

---

### See Also

*PM\_installServiceExt, PM\_startService, PM\_removeService*

## PM\_unloadDirectDraw

Unloads the DirectDraw libraries

### Declaration

```
void PMAPI PM_unloadDirectDraw(  
    int device)
```

### Prototype In

pmapi.h

### Parameters

*device*                      Index of the device to unload DirectDraw for (0 for primary)

### Description

Frees any DirectDraw objects for the device. We never actually explicitly unload the ddraw.dll library, since unloading and reloading it is unnecessary since we only want to unload it when the application exits and that happens automatically.

---

**Note:** *This function is Windows specific*

---

### See Also

*PM\_loadDirectDraw*

## PM\_unlockCodePages

Unlock code pages previously locked down.

### Declaration

```
int PMAPI PM_unlockCodePages(
    __codePtr p,
    uint len,
    PM_lockHandle *lh)
```

### Prototype In

pmapi.h

### Parameters

<i>p</i>	Linear pointer to the memory that was locked down
<i>len</i>	Length of the memory block that was locked down
<i>lh</i>	Pointer to the lock handle returned from <i>PM_lockCodePages</i>

### Description

This function is used to unlock a block of memory that was previously locked down with the *PM\_lockCodePages* function.

This version is used to unlock data pages in memory.

### See Also

*PM\_lockCodePages*

## PM\_unlockDataPages

Unlock data pages previously locked down.

### Declaration

```
int PMAPI PM_unlockDataPages(  
    void *p,  
    uint len,  
    PM_lockHandle *lh)
```

### Prototype In

pmapi.h

### Parameters

<i>p</i>	Linear pointer to the memory that was locked down
<i>len</i>	Length of the memory block that was locked down
<i>lh</i>	Pointer to the lock handle returned from <i>PM_lockDataPages</i>

### Description

This function is used to unlock a block of memory that was previously locked down with the *PM\_lockDataPages* function.

This version is used to unlock data pages in memory.

### See Also

*PM\_lockDataPages*

## PM\_useLocalMalloc

Use local memory allocation routines.

### Declaration

```
void PMAPI PM_useLocalMalloc(
    void * (*malloc)(size_t size),
    void * (*calloc)(size_t nelem, size_t size),
    void * (*realloc)(void *ptr, size_t size),
    void (*free)(void *p))
```

### Prototype In

pmapi.h

### Parameters

<i>malloc</i>	Pointer to new malloc routine to use
<i>calloc</i>	Pointer to new calloc routine to use
<i>realloc</i>	Pointer to new realloc routine to use
<i>free</i>	Pointer to new free routine to use

### Description

Tells the PM library to use a set of user specified memory allocation routines instead of using the normal malloc/calloc/realloc/free standard C library functions. This is useful if you wish to use a third party debugging malloc library or perhaps a set of faster memory allocation functions with the PM library, or any apps that use the PM library (such as the MGL). Once you have registered your memory allocation routines, all calls to *PM\_malloc*, *PM\_calloc*, *PM\_realloc* and *PM\_free* will be revector to your local memory allocation routines.

---

**Note:** *This function should be called right at the start of your application, before you initialise any other components or libraries.*

**Note:** *Code compiled into Binary Portable DLL's and Drivers automatically end up calling these functions via the BPD C runtime library.*

---

### See Also

*PM\_malloc, PM\_calloc, PM\_realloc, PM\_free*

## ULZElapsedTime

Compute the elapsed time between two timer counts.

### Declaration

```
ulong ZAPI ULZElapsedTime(  
    ulong start,  
    ulong finish)
```

### Prototype In

ztimer.h

### Parameters

<i>start</i>	Starting time for elapsed count
<i>finish</i>	Ending time for elapsed count

### Return Value

Elapsed timer in resolution counts.

### Description

Returns the elapsed time for the Ultra Long Period Zen Timer in units of the timers resolution (1/18th of a second under DOS). This function correctly computes the difference even if a midnight boundary has been crossed during the timing period.

### See Also

*ULZReadTime*, *ULZTimerResolution*

## ULZReadTime

Reads the current time from the Ultra Long Period Zen Timer.

### Declaration

```
ulong ZAPI ULZReadTime(void)
```

### Prototype In

ztimer.h

### Return Value

Current timer value in resolution counts.

### Description

Reads the current Ultra Long Period Zen Timer and returns it's current count. You can use the *ULZElapsedTime* function to find the elapsed time between two timer count readings.

### See Also

*ULZElapsedTime*, *ULZTimerResolution*



## ULZTimerCount

Returns the current count for the Ultra Long Period Zen Timer.

### Declaration

```
ulong ZAPI ULZTimerCount(void)
```

### Prototype In

ztimer.h

### Return Value

Count that has elapsed in resolution counts.

### Description

Returns the current count that has elapsed between calls to *ULZTimerOn* and *ULZTimerOff* in resolution counts.

### See Also

*ULZTimerOn*, *ULZTimerOff*, *ULZTimerLap*, *ULZTimerResolution*

## ULZTimerLap

Returns the current count for the Ultra Long Period Zen Timer and keeps it running.

### Declaration

```
ulong ZAPI ULZTimerLap(void)
```

### Prototype In

ztimer.h

### Return Value

Count that has elapsed in resolution counts.

### Description

Returns the current count that has elapsed since the last call to *ULZTimerOn* in microseconds. The time continues to run after this function is called so you can call this function repeatedly.

### See Also

*ULZTimerOn*, *ULZTimerOff*, *ULZTimerCount*

## ULZTimerOff

Stops the Long Period Zen Timer counting.

### Declaration

```
void ZAPI ULZTimerOff(void)
```

### Prototype In

ztimer.h

### Description

Stops the Ultra Long Period Zen Timer counting and latches the count. Once you have stopped the timer you can read the count with *ULZTimerCount*.

### See Also

*ULZTimerOn*, *ULZTimerLap*, *ULZTimerCount*

## ULZTimerOn

Starts the Ultra Long Period Zen Timer counting.

### Declaration

```
void ZAPI ULZTimerOn(void)
```

### Prototype In

ztimer.h

### Description

Starts the Ultra Long Period Zen Timer counting. Once you have started the timer, you can stop it with *ULZTimerOff* or you can latch the current count with *ULZTimerLap*.

The Ultra Long Period Zen Timer uses the available operating system services to obtain accurate timings results with as much precision as the operating system provides, but with enough granularity to time longer periods of time than the Long Period Zen Timer. Note that the resolution of the timer ticks is not constant between different platforms, and you should use the *ULZTimerResolution* function to determine the number of seconds in a single tick of the timer, and use this to convert the timer counts to seconds.

Under 32-bit Windows, we use the *timeGetTime* function which provides a resolution of 1 millisecond (0.001 of a second). Given that the timer count is returned as an unsigned 32-bit integer, this we can time intervals that are a maximum of  $2^{32}$  milliseconds in length (or about 1,200 hours or 50 days!).

Under 32-bit DOS, we use the system timer tick which runs at 18.2 times per second. Given that the timer count is returned as an unsigned 32-bit integer, this we can time intervals that are a maximum of  $2^{32} * (1/18.2)$  in length (or about 65,550 hours or 2731 days!).

### See Also

*ULZTimerOff*, *ULZTimerLap*, *ULZTimerCount*, *ULZElapsedTime*, *ULZReadTime*

## ULZTimerResolution

Returns the resolution of the Ultra Long Period Zen Timer.

### Declaration

```
void ZAPI ULZTimerResolution(  
    ulong *resolution)
```

### Prototype In

ztimer.h

### Parameters

*resolution*                      Place to store the timer in microseconds per timer count.

### Description

Returns the resolution of the Ultra Long Period Zen Timer as a 32-bit integer value measured in microseconds per timer count.

### See Also

*ULZReadTime, ULZElapsedTime, ULZTimerCount*

## ZTimerInit

Initializes the Zen Timer library.

### Declaration

```
void ZAPI ZTimerInit(void)
```

### Prototype In

ztimer.h

### Description

Obsolete function. Please use *ZTimerInitExt*.

## ZTimerInitExt

Initializes the Zen Timer library (extended)

### Declaration

```
void ZAPI ZTimerInitExt(  
    ibool accurate)
```

### Prototype In

ztimer.h

### Parameters

*accurate*                      True of the speed should be measured accurately

### Description

This function initializes the Zen Timer library, and *must* be called before any of the remaining Zen Timer library functions are called. The accurate parameter is used to determine whether highly accurate timing should be used or not. If high accuracy is needed, more time is spent profiling the actual speed of the CPU so that we can obtain highly accurate timing results, but the time spent in the initialisation routine will be significantly longer (on the order of 5 seconds).

## *Type Definitions*

---



## CPU\_largeInteger

### Declaration

```
typedef struct {  
    ulong    low;  
    ulong    high;  
} CPU_largeInteger
```

### Prototype In

cpuinfo.h

### Description

Defines the structure for holding 64-bit integers used for storing the values returned by the Intel RDTSC instruction.

### Members

<i>low</i>	Low 32-bits of the 64-bit integer
<i>high</i>	High 32-bits of the 64-bit integer

## CPU\_processorType

### Declaration

```
typedef enum {
    CPU_i386           = 0,
    CPU_i486           = 1,
    CPU_Pentium        = 2,
    CPU_PentiumPro     = 3,
    CPU_PentiumII      = 4,
    CPU_Celeron        = 5,
    CPU_PentiumIII     = 6,
    CPU_Pentium4       = 7,
    CPU_UnkIntel       = 8,
    CPU_Cyrix6x86      = 100,
    CPU_Cyrix6x86MX    = 101,
    CPU_CyrixMediaGX   = 102,
    CPU_CyrixMediaGXm  = 104,
    CPU_UnkCyrix       = 105,
    CPU_AMDAm486       = 200,
    CPU_AMDAm5x86      = 201,
    CPU_AMDK5          = 202,
    CPU_AMDK6          = 203,
    CPU_AMDK6_2        = 204,
    CPU_AMDK6_2plus    = 205,
    CPU_AMDK6_III      = 206,
    CPU_AMDK6_IIIplus  = 207,
    CPU_UnkAMD         = 208,
    CPU_AMDAthlon      = 250,
    CPU_AMDDuron       = 251,
    CPU_WinChipC6      = 300,
    CPU_WinChip2       = 301,
    CPU_UnkIDT         = 302,
    CPU_ViaCyrixIII    = 400,
    CPU_UnkVIA         = 401,
    CPU_Alpha          = 500,
    CPU_Mips           = 600,
    CPU_PowerPC        = 700,
    CPU_mask           = 0x00000FFF,
    CPU_IDT            = 0x00001000,
    CPU_Cyrix          = 0x00002000,
    CPU_AMD            = 0x00004000,
    CPU_Intel          = 0x00008000,
    CPU_VIA            = 0x00010000,
    CPU_familyMask     = 0x00FFF000,
    CPU_steppingMask   = 0x0F000000,
    CPU_steppingShift  = 24
} CPU_processorType
```

### Prototype In

cpuinfo.h

### Description

Defines the types of processors returned by *CPU\_getProcessorType*.

### Members

<i>CPU_i386</i>	Intel 80386 processor
<i>CPU_i486</i>	Intel 80486 processor
<i>CPU_Pentium</i>	Intel Pentium(R) processor

<i>CPU_PentiumPro</i>	Intel PentiumPro(R) processor
<i>CPU_PentiumII</i>	Intel PentiumII(R) processor
<i>CPU_Celeron</i>	Intel Celeron(R) processor
<i>CPU_PentiumIII</i>	Intel PentiumIII(R) processor
<i>CPU_Pentium4</i>	Intel Pentium4(R) processor
<i>CPU_UnkIntel</i>	Unknown Intel processor
<i>CPU_Cyrix6x86</i>	Cyrix 6x86 processor
<i>CPU_Cyrix6x86MX</i>	Cyrix 6x86MX processor
<i>CPU_CyrixMediaGX</i>	Cyrix MediaGX processor
<i>CPU_CyrixMediaGXm</i>	Cyrix MediaGXm processor
<i>CPU_UnkCyrix</i>	Unknown Cyrix processor
<i>CPU_AMDAm486</i>	AMD Am486 processor
<i>CPU_AMDAm5x86</i>	AMD Am5x86 processor
<i>CPU_AMDK5</i>	AMD K5 processor
<i>CPU_AMDK6</i>	AMD K6 processor
<i>CPU_AMDK6_2</i>	AMD K6-2 processor
<i>CPU_AMDK6_2plus</i>	AMD K6-2+ processor
<i>CPU_AMDK6_III</i>	AMD K6-III processor
<i>CPU_AMDK6_IIIplus</i>	AMD K6-III+ processor
<i>CPU_AMDAthlon</i>	AMD Athlon processor
<i>CPU_AMDDuron</i>	AMD Duron processor
<i>CPU_UnkAMD</i>	Unknown AMD processor
<i>CPU_WinChipC6</i>	IDT WinChip C6 processor
<i>CPU_WinChip2</i>	IDT WinChip 2 processor
<i>CPU_UnkIDT</i>	Unknown IDT processor
<i>CPU_ViaCyrixIII</i>	Via Cyrix III
<i>CPU_UnkVIA</i>	Unknown Via processor
<i>CPU_Alpha</i>	DEC Alpha processor
<i>CPU_Mips</i>	MIPS processor
<i>CPU_PowerPC</i>	PowerPC processor
<i>CPU_mask</i>	Mask to remove flags and get CPU type
<i>CPU_IDT</i>	This bit is set if the processor vendor is IDT
<i>CPU_Cyrix</i>	This bit is set if the processor vendor is Cyrix
<i>CPU_AMD</i>	This bit is set if the processor vendor is AMD
<i>CPU_Intel</i>	This bit is set if the processor vendor is Intel
<i>CPU_VIA</i>	This bit is set if the processor vendor is Via
<i>CPU_familyMask</i>	Mask to isolate CPU family
<i>CPU_steppingMask</i>	Mask to isolate CPU stepping
<i>CPU_steppingShift</i>	Shift factor for CPU stepping

## EVT\_asciiCodesType

### Declaration

```
typedef enum {
    ASCII_ctrlA      = 0x01,
    ASCII_ctrlB      = 0x02,
    ASCII_ctrlC      = 0x03,
    ASCII_ctrlD      = 0x04,
    ASCII_ctrlE      = 0x05,
    ASCII_ctrlF      = 0x06,
    ASCII_ctrlG      = 0x07,
    ASCII_backspace   = 0x08,
    ASCII_ctrlH      = 0x08,
    ASCII_tab         = 0x09,
    ASCII_ctrlI      = 0x09,
    ASCII_ctrlJ      = 0x0A,
    ASCII_ctrlK      = 0x0B,
    ASCII_ctrlL      = 0x0C,
    ASCII_enter       = 0x0D,
    ASCII_ctrlM      = 0x0D,
    ASCII_ctrlN      = 0x0E,
    ASCII_ctrlO      = 0x0F,
    ASCII_ctrlP      = 0x10,
    ASCII_ctrlQ      = 0x11,
    ASCII_ctrlR      = 0x12,
    ASCII_ctrlS      = 0x13,
    ASCII_ctrlT      = 0x14,
    ASCII_ctrlU      = 0x15,
    ASCII_ctrlV      = 0x16,
    ASCII_ctrlW      = 0x17,
    ASCII_ctrlX      = 0x18,
    ASCII_ctrlY      = 0x19,
    ASCII_ctrlZ      = 0x1A,
    ASCII_esc         = 0x1B,
    ASCII_space       = 0x20,
    ASCII_exclamation = 0x21,
    ASCII_quote       = 0x22,
    ASCII_pound       = 0x23,
    ASCII_dollar      = 0x24,
    ASCII_percent     = 0x25,
    ASCII_ampersand   = 0x26,
    ASCII_apostrophe  = 0x27,
    ASCII_leftBrace   = 0x28,
    ASCII_rightBrace  = 0x29,
    ASCII_times       = 0x2A,
    ASCII_plus        = 0x2B,
    ASCII_comma       = 0x2C,
    ASCII_minus       = 0x2D,
    ASCII_period      = 0x2E,
    ASCII_divide      = 0x2F,
    ASCII_0           = 0x30,
    ASCII_1           = 0x31,
    ASCII_2           = 0x32,
    ASCII_3           = 0x33,
    ASCII_4           = 0x34,
    ASCII_5           = 0x35,
    ASCII_6           = 0x36,
    ASCII_7           = 0x37,
    ASCII_8           = 0x38,
    ASCII_9           = 0x39,
    ASCII_colon       = 0x3A,
```

ASCII_semicolon	= 0x3B,
ASCII_lessThan	= 0x3C,
ASCII_equals	= 0x3D,
ASCII_greaterThan	= 0x3E,
ASCII_question	= 0x3F,
ASCII_at	= 0x40,
ASCII_A	= 0x41,
ASCII_B	= 0x42,
ASCII_C	= 0x43,
ASCII_D	= 0x44,
ASCII_E	= 0x45,
ASCII_F	= 0x46,
ASCII_G	= 0x47,
ASCII_H	= 0x48,
ASCII_I	= 0x49,
ASCII_J	= 0x4A,
ASCII_K	= 0x4B,
ASCII_L	= 0x4C,
ASCII_M	= 0x4D,
ASCII_N	= 0x4E,
ASCII_O	= 0x4F,
ASCII_P	= 0x50,
ASCII_Q	= 0x51,
ASCII_R	= 0x52,
ASCII_S	= 0x53,
ASCII_T	= 0x54,
ASCII_U	= 0x55,
ASCII_V	= 0x56,
ASCII_W	= 0x57,
ASCII_X	= 0x58,
ASCII_Y	= 0x59,
ASCII_Z	= 0x5A,
ASCII_leftSquareBrace	= 0x5B,
ASCII_backSlash	= 0x5C,
ASCII_rightSquareBrace	= 0x5D,
ASCII_caret	= 0x5E,
ASCII_underscore	= 0x5F,
ASCII_leftApostrophe	= 0x60,
ASCII_a	= 0x61,
ASCII_b	= 0x62,
ASCII_c	= 0x63,
ASCII_d	= 0x64,
ASCII_e	= 0x65,
ASCII_f	= 0x66,
ASCII_g	= 0x67,
ASCII_h	= 0x68,
ASCII_i	= 0x69,
ASCII_j	= 0x6A,
ASCII_k	= 0x6B,
ASCII_l	= 0x6C,
ASCII_m	= 0x6D,
ASCII_n	= 0x6E,
ASCII_o	= 0x6F,
ASCII_p	= 0x70,
ASCII_q	= 0x71,
ASCII_r	= 0x72,
ASCII_s	= 0x73,
ASCII_t	= 0x74,
ASCII_u	= 0x75,
ASCII_v	= 0x76,
ASCII_w	= 0x77,
ASCII_x	= 0x78,
ASCII_y	= 0x79,

```
ASCII_z           = 0x7A,  
ASCII_leftCurlyBrace = 0x7B,  
ASCII_verticalBar  = 0x7C,  
ASCII_rightCurlyBrace = 0x7D,  
ASCII_tilde       = 0x7E  
} EVT_asciiCodesType
```

### Prototype In event.h

### Description

Defines the set of ASCII codes reported by the event library functions in the message field. Use the *EVT\_asciiCode* macro to extract the code from the event structure.

## EVT\_eventJoyAxisType

### Declaration

```
typedef enum {
    EVT_JOY_AXIS_X1      = 0x00000001,
    EVT_JOY_AXIS_Y1      = 0x00000002,
    EVT_JOY_AXIS_X2      = 0x00000004,
    EVT_JOY_AXIS_Y2      = 0x00000008,
    EVT_JOY_AXIS_ALL     = 0x0000000F
} EVT_eventJoyAxisType
```

### Prototype In

event.h

### Description

Defines the mask for the joystick axes that are present

### Members

<i>EVT_JOY_AXIS_X1</i>	Joystick 1, X axis is present
<i>EVT_JOY_AXIS_Y1</i>	Joystick 1, Y axis is present
<i>EVT_JOY_AXIS_X2</i>	Joystick 2, X axis is present
<i>EVT_JOY_AXIS_Y2</i>	Joystick 2, Y axis is present
<i>EVT_JOY_AXIS_ALL</i>	Mask for all axes

## EVT\_eventJoyMaskType

### Declaration

```
typedef enum {  
    EVT_JOY1_BUTTONA    = 0x00000001,  
    EVT_JOY1_BUTTONB    = 0x00000002,  
    EVT_JOY2_BUTTONA    = 0x00000004,  
    EVT_JOY2_BUTTONB    = 0x00000008  
} EVT_eventJoyMaskType
```

### Prototype In

event.h

### Description

Defines the event message masks for joystick events

### Members

<i>EVT_JOY1_BUTTONA</i>	Joystick 1, button A is down
<i>EVT_JOY1_BUTTONB</i>	Joystick 1, button B is down
<i>EVT_JOY2_BUTTONA</i>	Joystick 2, button A is down
<i>EVT_JOY2_BUTTONB</i>	Joystick 2, button B is down



## EVT\_eventMaskType

### Declaration

```
typedef enum {
    EVT_KEYEVT      = (EVT_KEYDOWN | EVT_KEYREPEAT | EVT_KEYUP),
    EVT_MOUSEEVT    = (EVT_MOUSEDOWN | EVT_MOUSEAUTO | EVT_MOUSEUP |
EVT_MOUSEMOVE),
    EVT_MOUSECLICK  = (EVT_MOUSEDOWN | EVT_MOUSEUP),
    EVT_JOYEVT      = (EVT_JOYCLICK | EVT_JOYMOVE),
    EVT_EVERYEVT    = 0x7FFFFFFF
} EVT_eventMaskType
```

### Prototype In

event.h

### Description

Defines the event code masks you can use to test for multiple types of events, since the event codes are mutually exclusive bit fields.

### Members

<i>EVT_KEYEVT</i>	Mask for any key event
<i>EVT_MOUSEEVT</i>	Mask for any mouse event
<i>EVT_MOUSECLICK</i>	Mask for any mouse click event
<i>EVT_JOYEVT</i>	Mask for any joystick event
<i>EVT_EVERYEVT</i>	Mask for any event

## EVT\_eventModMaskType

### Declaration

```
typedef enum {
    EVT_LEFTBUT      = 0x00000001,
    EVT_RIGHTBUT     = 0x00000002,
    EVT_MIDDLEBUT    = 0x00000004,
    EVT_RIGHTSHIFT   = 0x00000008,
    EVT_LEFTSHIFT    = 0x00000010,
    EVT_RIGHTCTRL    = 0x00000020,
    EVT_RIGHTALT     = 0x00000040,
    EVT_LEFTCTRL     = 0x00000080,
    EVT_LEFTALT      = 0x00000100,
    EVT_SHIFTKEY     = 0x00000018,
    EVT_CTRLSTATE    = 0x000000A0,
    EVT_ALTSTATE     = 0x00000140,
    EVT_SCROLLLOCK   = 0x00000200,
    EVT_NUMLOCK      = 0x00000400,
    EVT_CAPSLOCK     = 0x00000800
} EVT_eventModMaskType
```

### Prototype In event.h

### Description

Defines the event modifier masks. These are the masks used to extract the modifier information from the modifiers field of the *event\_t* structure. Note that the values in the modifiers field represent the values of these modifier keys at the time the event occurred, not the time you decided to process the event.

### Members

<b><i>EVT_LEFTBUT</i></b>	Set if left mouse button was down
<b><i>EVT_RIGHTBUT</i></b>	Set if right mouse button was down
<b><i>EVT_MIDDLEBUT</i></b>	Set if the middle button was down
<b><i>EVT_RIGHTSHIFT</i></b>	Set if right shift was down
<b><i>EVT_LEFTSHIFT</i></b>	Set if left shift was down
<b><i>EVT_RIGHTCTRL</i></b>	Set if right ctrl key was down
<b><i>EVT_RIGHTALT</i></b>	Set if right alt key was down
<b><i>EVT_LEFTCTRL</i></b>	Set if left ctrl key was down
<b><i>EVT_LEFTALT</i></b>	Set if left alt key was down
<b><i>EVT_SHIFTKEY</i></b>	Mask for any shift key down
<b><i>EVT_CTRLSTATE</i></b>	Set if ctrl key was down
<b><i>EVT_ALTSTATE</i></b>	Set if alt key was down
<b><i>EVT_CAPSLOCK</i></b>	Caps lock is active
<b><i>EVT_NUMLOCK</i></b>	Num lock is active
<b><i>EVT_SCROLLLOCK</i></b>	Scroll lock is active

## EVT\_eventMouseMaskType

### Declaration

```
typedef enum {
    EVT_LEFTBMask    = 0x00000001,
    EVT_RIGHTBMask   = 0x00000002,
    EVT_MIDDLEBMask  = 0x00000004,
    EVT_BOTHBMask    = 0x00000007,
    EVT_ALLBMask     = 0x00000007,
    EVT_DBLCLICK     = 0x00010000
} EVT_eventMouseMaskType
```

### Prototype In

event.h

### Description

Defines the event message masks for mouse events

### Members

<i>EVT_LEFTBMask</i>	Left button is held down
<i>EVT_RIGHTBMask</i>	Right button is held down
<i>EVT_MIDDLEBMask</i>	Middle button is held down
<i>EVT_BOTHBMask</i>	Both left and right held down together
<i>EVT_ALLBMask</i>	All buttons pressed
<i>EVT_DBLCLICK</i>	Set if mouse down event was a double click

## EVT\_eventType

### Declaration

```
typedef enum {
    EVT_NULLEVT      = 0x00000000,
    EVT_KEYDOWN      = 0x00000001,
    EVT_KEYREPEAT    = 0x00000002,
    EVT_KEYUP        = 0x00000004,
    EVT_MOUSEDOWN    = 0x00000008,
    EVT_MOUSEAUTO    = 0x00000010,
    EVT_MOUSEUP      = 0x00000020,
    EVT_MOUSEMOVE    = 0x00000040,
    EVT_JOYCLICK     = 0x00000080,
    EVT_JOYMOVE      = 0x00000100,
    EVT_USEREVT      = 0x00000200
} EVT_eventType
```

### Prototype In event.h

### Description

Defines the event codes returned in the *event\_t* structures what field. Note that these are defined as a set of mutually exclusive bit fields, so you can test for multiple event types using the combined event masks defined in the *EVT\_eventMaskType* enumeration.

### Members

<i>EVT_NULLEVT</i>	A null event
<i>EVT_KEYDOWN</i>	Key down event
<i>EVT_KEYREPEAT</i>	Key repeat event
<i>EVT_KEYUP</i>	Key up event
<i>EVT_MOUSEDOWN</i>	Mouse down event
<i>EVT_MOUSEAUTO</i>	Mouse down autorepeat event
<i>EVT_MOUSEUP</i>	Mouse up event
<i>EVT_MOUSEMOVE</i>	Mouse movement event
<i>EVT_JOYCLICK</i>	Joystick button state change event
<i>EVT_JOYMOVE</i>	Joystick movement event
<i>EVT_USEREVT</i>	First user event

## EVT\_masksType

### Declaration

```
typedef enum {  
    EVT_ASCII_MASK    = 0x00FF,  
    EVT_SCANMASK      = 0xFF00,  
    EVT_COUNTMASK     = 0x7FFF0000L  
} EVT_masksType
```

### Prototype In

event.h

### Description

Defines the event message masks used to extract information for keyboard events

### Members

<i>EVT_ASCII_MASK</i>	ASCII code of key pressed
<i>EVT_SCANMASK</i>	Scan code of key pressed
<i>EVT_COUNTMASK</i>	Count for KEYREPEAT's

## EVT\_scanCodesType

### Declaration

```
typedef enum {
    KB_padEnter           = 0x60,
    KB_padMinus           = 0x4A,
    KB_padPlus            = 0x4E,
    KB_padTimes           = 0x37,
    KB_padDivide          = 0x61,
    KB_padLeft            = 0x62,
    KB_padRight           = 0x63,
    KB_padUp              = 0x64,
    KB_padDown            = 0x65,
    KB_padInsert          = 0x66,
    KB_padDelete          = 0x67,
    KB_padHome            = 0x68,
    KB_padEnd             = 0x69,
    KB_padPageUp          = 0x6A,
    KB_padPageDown       = 0x6B,
    KB_padCenter          = 0x4C,
    KB_F1                 = 0x3B,
    KB_F2                 = 0x3C,
    KB_F3                 = 0x3D,
    KB_F4                 = 0x3E,
    KB_F5                 = 0x3F,
    KB_F6                 = 0x40,
    KB_F7                 = 0x41,
    KB_F8                 = 0x42,
    KB_F9                 = 0x43,
    KB_F10                = 0x44,
    KB_F11                = 0x57,
    KB_F12                = 0x58,
    KB_left               = 0x4B,
    KB_right              = 0x4D,
    KB_up                 = 0x48,
    KB_down               = 0x50,
    KB_insert             = 0x52,
    KB_delete             = 0x53,
    KB_home               = 0x47,
    KB_end                = 0x4F,
    KB_pageUp             = 0x49,
    KB_pageDown           = 0x51,
    KB_capsLock           = 0x3A,
    KB_numLock            = 0x45,
    KB_scrollLock         = 0x46,
    KB_leftShift          = 0x2A,
    KB_rightShift         = 0x36,
    KB_leftCtrl           = 0x1D,
    KB_rightCtrl          = 0x6C,
    KB_leftAlt            = 0x38,
    KB_rightAlt           = 0x6D,
    KB_leftWindows        = 0x5B,
    KB_rightWindows       = 0x5C,
    KB_menu               = 0x5D,
    KB_sysReq             = 0x54,
    KB_esc                = 0x01,
    KB_1                  = 0x02,
    KB_2                  = 0x03,
    KB_3                  = 0x04,
    KB_4                  = 0x05,
    KB_5                  = 0x06,
```

```

KB_6           = 0x07,
KB_7           = 0x08,
KB_8           = 0x09,
KB_9           = 0x0A,
KB_0           = 0x0B,
KB_minus       = 0x0C,
KB_equals      = 0x0D,
KB_backSlash   = 0x2B,
KB_backspace   = 0x0E,
KB_tab         = 0x0F,
KB_Q           = 0x10,
KB_W           = 0x11,
KB_E           = 0x12,
KB_R           = 0x13,
KB_T           = 0x14,
KB_Y           = 0x15,
KB_U           = 0x16,
KB_I           = 0x17,
KB_O           = 0x18,
KB_P           = 0x19,
KB_leftSquareBrace = 0x1A,
KB_rightSquareBrace = 0x1B,
KB_enter       = 0x1C,
KB_A           = 0x1E,
KB_S           = 0x1F,
KB_D           = 0x20,
KB_F           = 0x21,
KB_G           = 0x22,
KB_H           = 0x23,
KB_J           = 0x24,
KB_K           = 0x25,
KB_L           = 0x26,
KB_semicolon   = 0x27,
KB_apostrophe  = 0x28,
KB_Z           = 0x2C,
KB_X           = 0x2D,
KB_C           = 0x2E,
KB_V           = 0x2F,
KB_B           = 0x30,
KB_N           = 0x31,
KB_M           = 0x32,
KB_comma       = 0x33,
KB_period      = 0x34,
KB_divide      = 0x35,
KB_space       = 0x39,
KB_tilde       = 0x29
} EVT_scanCodesType

```

## Prototype In event.h

### Description

Defines the set of scan codes reported by the event library functions in the message field. Use the *EVT\_scanCode* macro to extract the code from the event structure. Note that the scan codes reported will be the same across all keyboards (assuming the placement of keys on a 101 key US keyboard), but the translated ASCII values may be different depending on the country code pages in use.

---

**Note:** *Scan codes in the event library are not really hardware scan codes, but rather virtual scan codes as generated by a low level keyboard interface driver. All virtual codes begin with scan code 0x60 and range up from there.*

---



## LZTimerObject

### Declaration

```
typedef struct {  
    CPU_largeInteger    start;  
    CPU_largeInteger    end;  
} LZTimerObject
```

### Prototype In

ztimer.h

### Description

Defines the structure for an *LZTimerObject* which contains the starting and ending timestamps for the timer. By putting the timer information into a structure the Zen Timer can be used for multiple timers running simultaneously.

### Members

<i>start</i>	Starting 64-bit timer count
<i>end</i>	Ending 64-bit timer count

## PCIAGPCapability

### Declaration

```
typedef struct {  
    PCICapsHeader    h;  
    ushort           majMin;  
    PCIAGPStatus      AGPStatus;  
    PCIAGPCommand     AGPCommand;  
} PCIAGPCapability
```

### Prototype In

pcilib.h

### Description

AGP Capability structure

### Members

<i>h</i>	PCI capabilities header block
<i>majMin</i>	Major/minor number
<i>AGPStatus</i>	AGP status field
<i>AGPCommand</i>	AGP command field

## PCIAGPCommand

### Declaration

```
typedef struct {
    uint    rate:4;
    uint    fastWriteEnable:1;
    uint    fourGBEnable:1;
    uint    rsvd1:2;
    uint    AGPEnable:1;
    uint    SBAEnable:1;
    uint    rsvd2:14;
    uint    requestQueueDepth:8;
} PCIAGPCommand
```

### Prototype In

pcilib.h

### Description

Structure defining the PCI AGP command register contents

### Members

<i>rate</i>	Enable AGP rate (1-8x)
<i>fastWriteEnable</i>	Enable AGP FastWrite
<i>fourGBEnable</i>	Enable 4GB addressing
<i>rsvd1</i>	Reserved; not used
<i>AGPEnable</i>	Enable AGP bus
<i>SBAEnable</i>	Enable side band addressing
<i>rsvd2</i>	Reserved; not used
<i>requestQueueDepth</i>	Request queue depth

## PCIAGPStatus

### Declaration

```
typedef struct {
    uint    rate:4;
    uint    fastWrite:1;
    uint    fourGB:1;
    uint    rsvd1:3;
    uint    sideBandAddressing:1;
    uint    rsvd2:14;
    uint    requestQueueDepthMaximum:8;
} PCIAGPStatus
```

### Prototype In

pcilib.h

### Description

Structure defining the PCI AGP status register contents

### Members

<i>rate</i>	Supported AGP rate (1-8x)
<i>fastWrite</i>	AGP FastWrite supported
<i>fourGB</i>	4GB addressing supported
<i>rsvd1</i>	Reserved; not used
<i>sideBandAddressing</i>	Side band addressing supported
<i>rsvd2</i>	Reserved; not used
<i>requestQueueDepthMaximum</i>	Maximum request queue depth

## PCIAccessRegFlags

### Declaration

```
typedef enum {  
    PCI_READ_BYTE           = 0,  
    PCI_READ_WORD           = 1,  
    PCI_READ_DWORD          = 2,  
    PCI_WRITE_BYTE          = 3,  
    PCI_WRITE_WORD          = 4,  
    PCI_WRITE_DWORD         = 5  
} PCIAccessRegFlags
```

### Prototype In

pcilib.h

### Description

Function codes to pass to *PCI\_accessReg*. The names of the flags are self explanatory.

## PCICapsHeader

### Declaration

```
typedef struct {  
    uchar    capsID;  
    uchar    next;  
} PCICapsHeader
```

### Prototype In

pcilib.h

### Description

PCI Capability header structure. All PCI capabilities have this header field to describe the capability type.

### Members

<i>capsID</i>	Used to identify the type of the structure ( <i>PCICapsType</i> ).
<i>next</i>	Next is the offset in PCI configuration space (0x40-0xFC) of the next capability structure in the list, or 0x00 if there are no more entries.

## PCICapsType

### Declaration

```
typedef enum {  
    PCI_capsPowerManagement    = 0x01,  
    PCI_capsAGP                 = 0x02,  
    PCI_capsMSI                 = 0x05  
} PCICapsType
```

### Prototype In

pcilib.h

### Description

Defines for the PCI capability IDs, which define what extended PCI capabilities a device supports. The names of the flags are self explanatory.

## PCIClassTypes

### Declaration

```
typedef enum {
    PCI_BRIDGE_CLASS          = 0x06,
    PCI_HOST_BRIDGE_SUBCLASS  = 0x00,
    PCI_EARLY_VGA_CLASS       = 0x00,
    PCI_EARLY_VGA_SUBCLASS    = 0x01,
    PCI_DISPLAY_CLASS         = 0x03,
    PCI_DISPLAY_VGA_SUBCLASS   = 0x00,
    PCI_DISPLAY_XGA_SUBCLASS   = 0x01,
    PCI_DISPLAY_OTHER_SUBCLASS = 0x80,
    PCI_MM_CLASS              = 0x04,
    PCI_AUDIO_SUBCLASS         = 0x01
} PCIClassTypes
```

### Prototype In

pcilib.h

### Description

Defines for the known PCI class device types and sub class device types. The names of the types are self explanatory.



## PCICommandFlags

### Declaration

```
typedef enum {
    PCI_enableIOspace           = 0x0001,
    PCI_enableMemorySpace      = 0x0002,
    PCI_enableBusMaster         = 0x0004,
    PCI_enableSpecialCycles     = 0x0008,
    PCI_enableWriteAndInvalidate = 0x0010,
    PCI_enableVGACompatiblePalette = 0x0020,
    PCI_enableParity            = 0x0040,
    PCI_enableWaitCycle         = 0x0080,
    PCI_enableSerr              = 0x0100,
    PCI_enableFastBackToBack    = 0x0200
} PCICommandFlags
```

### Prototype In

pcilib.h

### Description

Defines for the *PCIDeviceInfo.Command* field, which control which features of the device are enabled or disabled. A 1 in the field means that the feature is enabled, 0 means it is disabled. The names of the flags are self explanatory.

## PCIDeviceInfo

### Declaration

```
typedef struct {
    ulong          dwSize;
    PCISlot        slot;
    ulong          mechl;
    ushort         VendorID;
    ushort         DeviceID;
    ushort         Command;
    ushort         Status;
    uchar          RevID;
    uchar          Interface;
    uchar          SubClass;
    uchar          BaseClass;
    uchar          CacheLineSize;
    uchar          LatencyTimer;
    uchar          HeaderType;
    uchar          BIST;
    union {
        PCIType0Info  type0;
        PCIType1Info  type1;
        PCIType2Info  type2;
    } u;
} PCIDeviceInfo
```

### Prototype In

pcilib.h

### Description

Structure defining the PCI configuration space information for a single PCI device on the PCI bus. We enumerate all this information for all PCI devices on the bus.

---

**Note:** The *dwSize* member is intended for future compatibility, and should be set to the size of the structure as defined in this header file. Future drivers will be compatible with older software by examining this value.

---

### Members

<i>dwSize</i>	Set to size of structure in bytes
<i>slot</i>	PCI slot identifier for this device
<i>mechl</i>	True if we enumerated this bus using PCI access mechanism 1
<i>VendorID</i>	Unique PCI device Vendor ID value
<i>DeviceID</i>	Unique PCI device Device ID value
<i>Command</i>	Device command register used to control the device ( <i>PCICCommandFlags</i> )
<i>Status</i>	Device status register flags ( <i>PCIStatusFlags</i> )
<i>RevID</i>	Device revision ID value
<i>Interface</i>	Device interface type value
<i>SubClass</i>	Device Sub Class field
<i>BaseClass</i>	Device Base Class field
<i>CacheLineSize</i>	Cache line size for the device
<i>LatencyTimer</i>	Latency timer value

<i>HeaderType</i>	Header type field, defining type of info ( <i>PCIHeaderTypeFlags</i> )
<i>BIST</i>	BIST value
<i>type0</i>	Union to access PCI type 0 specific information
<i>type1</i>	Union to access PCI type 1 specific information
<i>type2</i>	Union to access PCI type 2 specific information

## PCIHeaderTypeFlags

### Declaration

```
typedef enum {  
    PCI_deviceType           = 0x00,  
    PCI_bridgeType           = 0x01,  
    PCI_cardBusBridgeType    = 0x02,  
    PCI_multiFunctionType    = 0x80  
} PCIHeaderTypeFlags
```

### Prototype In

pcilib.h

### Description

Defines for the *PCIDeviceInfo*.HeaderType field. The names of the flags are self explanatory.

## PCIStatusFlags

### Declaration

```
typedef enum {
    PCI_statusCapabilitiesList      = 0x0010,
    PCI_status66MhzCapable         = 0x0020,
    PCI_statusUDFSupported         = 0x0040,
    PCI_statusFastBackToBack       = 0x0080,
    PCI_statusDataParityDetected   = 0x0100,
    PCI_statusDevSel               = 0x0600,
    PCI_statusSignaledTargetAbort  = 0x0800,
    PCI_statusRecievedTargetAbort  = 0x1000,
    PCI_statusRecievedMasterAbort  = 0x2000,
    PCI_statusSignaledSystemError  = 0x4000,
    PCI_statusDetectedParityError  = 0x8000
} PCIStatusFlags
```

### Prototype In

pcilib.h

### Description

Defines for the *PCIDeviceInfo*.Status field, which control which features of the device are supported. A 1 in the field means that the feature is supported, 0 means it is not supported. The names of the flags are self explanatory.

## PCType0Info

### Declaration

```
typedef struct {
    ulong    BaseAddress10;
    ulong    BaseAddress14;
    ulong    BaseAddress18;
    ulong    BaseAddress1C;
    ulong    BaseAddress20;
    ulong    BaseAddress24;
    ulong    CardbusCISPointer;
    ushort   SubSystemVendorID;
    ushort   SubSystemID;
    ulong    ROMBaseAddress;
    uchar    CapabilitiesPointer;
    uchar    reserved1;
    uchar    reserved2;
    uchar    reserved3;
    ulong    reserved4;
    uchar    InterruptLine;
    uchar    InterruptPin;
    uchar    MinimumGrant;
    uchar    MaximumLatency;
    ulong    BaseAddress10Len;
    ulong    BaseAddress14Len;
    ulong    BaseAddress18Len;
    ulong    BaseAddress1CLen;
    ulong    BaseAddress20Len;
    ulong    BaseAddress24Len;
    ulong    ROMBaseAddressLen;
} PCType0Info
```

### Prototype In

pcilib.h

### Description

Structure defining the regular (type 0) PCI configuration register layout. We use this in the *PCIDeviceInfo* union so we can describe all types of PCI configuration spaces with a single structure.

---

**Note:** *The PCI base address length values are not actually in the PCI configuration space, but are calculated when the configuration space is enumerated as they are useful values to know.*

---

### Members

<i>BaseAddress10</i>	Base address register (BAR) 10h
<i>BaseAddress14</i>	Base address register (BAR) 14h
<i>BaseAddress18</i>	Base address register (BAR) 18h
<i>BaseAddress1C</i>	Base address register (BAR) 1Ch
<i>BaseAddress20</i>	Base address register (BAR) 20h
<i>BaseAddress24</i>	Base address register (BAR) 24h
<i>CardbusCISPointer</i>	Pointer to CardBus Information Structure in config space
<i>SubSystemVendorID</i>	Sub System Vendor ID for this device type

<i>SubSystemID</i>	Sub System ID for this device type
<i>ROMBaseAddress</i>	Base address for ROM on device (if any)
<i>CapabilitiesPointer</i>	Pointer to PCI capabilities list
<i>reserved1</i>	Reserved: not used for this device type
<i>reserved2</i>	Reserved: not used for this device type
<i>reserved3</i>	Reserved: not used for this device type
<i>reserved4</i>	Reserved: not used for this device type
<i>InterruptLine</i>	Interrupt line assigned to this device
<i>InterruptPin</i>	Interrupt pin assigned to this device
<i>MinimumGrant</i>	Minimum interrupt grant assigned to this device
<i>MaximumLatency</i>	Maximum interrupt latency assigned to this device
<i>BaseAddress10Len</i>	Length of BAR 10 (calculated value)
<i>BaseAddress14Len</i>	Length of BAR 14 (calculated value)
<i>BaseAddress18Len</i>	Length of BAR 18 (calculated value)
<i>BaseAddress1CLen</i>	Length of BAR 1C (calculated value)
<i>BaseAddress20Len</i>	Length of BAR 20 (calculated value)
<i>BaseAddress24Len</i>	Length of BAR 24 (calculated value)
<i>ROMBaseAddressLen</i>	Length of ROM (calculated value)

## PCIType1Info

### Declaration

```
typedef struct {
    ulong    BaseAddress10;
    ulong    BaseAddress14;
    uchar    PrimaryBusNumber;
    uchar    SecondaryBusNumber;
    uchar    SubordinateBusNumber;
    uchar    SecondaryLatencyTimer;
    uchar    IOBase;
    uchar    IOLimit;
    ushort   SecondaryStatus;
    ushort   MemoryBase;
    ushort   MemoryLimit;
    ushort   PrefetchableMemoryBase;
    ushort   PrefetchableMemoryLimit;
    ulong    PrefetchableBaseHi;
    ulong    PrefetchableLimitHi;
    ushort   IOBaseHi;
    ushort   IOLimitHi;
    uchar    CapabilitiesPointer;
    uchar    reserved1;
    uchar    reserved2;
    uchar    reserved3;
    ulong    ROMBaseAddress;
    uchar    InterruptLine;
    uchar    InterruptPin;
    ushort   BridgeControl;
} PCIType1Info
```

### Prototype In pcilib.h

### Description

Structure defining PCI to PCI bridge (type 1) PCI configuration register layout. We use this in the *PCIDeviceInfo* union so we can describe all types of PCI configuration spaces with a single structure.

### Members

<i>BaseAddress10</i>	Base address register (BAR) 10h
<i>BaseAddress14</i>	Base address register (BAR) 14h
<i>PrimaryBusNumber</i>	Primary bus number this bridge lives on
<i>SecondaryBusNumber</i>	Secondary bus number this bridge controls
<i>SubordinateBusNumber</i>	Subordinate bus number for this bridge
<i>SecondaryLatencyTimer</i>	Secondary latency timer
<i>IOBase</i>	I/O base address for bridge control registers
<i>IOLimit</i>	I/O limit for bridge control registers
<i>SecondaryStatus</i>	Secondary status
<i>MemoryBase</i>	Memory mapped base address for bridge control registers
<i>MemoryLimit</i>	Memory mapped limit for bridge control registers
<i>PrefetchableMemoryBase</i>	Base of pre-fetchable memory on bus



<i>PrefetchableMemoryLimit</i>	Length of pre-fetchable memory on bus
<i>PrefetchableBaseHi</i>	High portion of prefetchable base value
<i>PrefetchableLimitHi</i>	High portion of prefetchable limit value
<i>IOBaseHi</i>	High value of I/O base address
<i>IOLimitHi</i>	High value of I/O limit
<i>CapabilitiesPointer</i>	Pointer to PCI bridge capabilities list
<i>reserved1</i>	Reserved: not used for this device type
<i>reserved2</i>	Reserved: not used for this device type
<i>reserved3</i>	Reserved: not used for this device type
<i>ROMBaseAddress</i>	Address if ROM for bridge (if any)
<i>InterruptLine</i>	Interrupt line assigned to this device
<i>InterruptPin</i>	Interrupt pin assigned to this device
<i>BridgeControl</i>	Bridge control register

## PCType2Info

### Declaration

```
typedef struct {
    ulong    SocketRegistersBaseAddress;
    uchar    CapabilitiesPointer;
    uchar    reserved1;
    ushort   SecondaryStatus;
    uchar    PrimaryBus;
    uchar    SecondaryBus;
    uchar    SubordinateBus;
    uchar    SecondaryLatency;
    struct {
        ulong    Base;
        ulong    Limit;
    } Range[4];
    uchar    InterruptLine;
    uchar    InterruptPin;
    ushort   BridgeControl;
} PCType2Info
```

### Prototype In

pcilib.h

### Description

Structure defining PCI to CardBus bridge (type 2) PCI configuration register layout. We use this in the *PCIDeviceInfo* union so we can describe all types of PCI configuration spaces with a single structure.

### Members

<i>SocketRegistersBaseAddress</i>	Base address for control registers
<i>CapabilitiesPointer</i>	CardBus bridge capabilities pointer
<i>reserved1</i>	Reserved: not used for this device type
<i>SecondaryStatus</i>	Secondary status
<i>PrimaryBus</i>	Primary bus number bridge is connected to
<i>SecondaryBus</i>	Secondary bus bridge controls
<i>SubordinateBus</i>	Subordinate bus for bridge
<i>SecondaryLatency</i>	Secondary latency
<i>Range</i>	Array of four base/limit ranges
<i>InterruptLine</i>	Interrupt line assigned to this device
<i>InterruptPin</i>	Interrupt pin assigned to this device
<i>BridgeControl</i>	Bridge control register

## PCIslot

### Declaration

```
typedef union {
    struct {
        uint    Zero:2;
        uint    Register:6;
        uint    Function:3;
        uint    Device:5;
        uint    Bus:8;
        uint    Reserved:7;
        uint    Enable:1;
    } p;
    ulong    i;
} PCIslot
```

### Prototype In

pcilib.h

### Description

Structure defining a PCI slot identifier.

---

**Note:** *We define all bitfield's as uint's, specifically so that the IBM Visual Age C++ compiler does not complain. We need them to be 32-bits wide, and this is the width of an unsigned integer, but we can't use a ulong to make this explicit or we get errors.*

---

### Members

<i>Zero</i>	Always set to zero
<i>Register</i>	Field containing the PCI register index
<i>Function</i>	Field containing the PCI function index
<i>Device</i>	Field containing the PCI device index
<i>Bus</i>	Field containing the PCI bus index
<i>Reserved</i>	Reserved (always 0)
<i>Enable</i>	Enable bit to enable this slot
<i>i</i>	Union field to access as a 32-bit integer

## PE\_errorCodes

### Declaration

```
typedef enum {
    PE_ok,
    PE_fileNotFound,
    PE_outOfMemory,
    PE_invalidDLLImage,
    PE_unableToInitLibC,
    PE_unknownImageFormat
} PE_errorCodes
```

### Prototype In

drvlib/peloder.h

### Description

Defines the error codes returned by the library

### Members

<i>PE_ok</i>	No error
<i>PE_fileNotFound</i>	DLL file not found
<i>PE_outOfMemory</i>	Out of memory loading DLL
<i>PE_invalidDLLImage</i>	DLL image is invalid or corrupted
<i>PE_unableToInitLibC</i>	Unable to initialise the C runtime library
<i>PE_unknownImageFormat</i>	DLL image is in a format that is not supported

## PMBYTEREGS

### Declaration

```
typedef struct {  
    uchar    al;  
    uchar    ah; ushort ax_hi;  
    uchar    bl;  
    uchar    bh; ushort bx_hi;  
    uchar    cl;  
    uchar    ch; ushort cx_hi;  
    uchar    dl;  
    uchar    dh; ushort dx_hi;  
} PMBYTEREGS
```

### Prototype In

pmapi.h

### Description

Structure describing the 8-bit x86 CPU registers

### Members

<i>al</i>	Value of the AL register
<i>ah</i>	Value of the AH register
<i>bl</i>	Value of the BL register
<i>bh</i>	Value of the BH register
<i>cl</i>	Value of the CL register
<i>ch</i>	Value of the CH register
<i>dl</i>	Value of the DL register
<i>dh</i>	Value of the DH register

## PMDWORDREGS

### Declaration

```
typedef struct {  
    ulong    eax;  
    ulong    ebx;  
    ulong    ecx;  
    ulong    edx;  
    ulong    esi;  
    ulong    edi;  
    ulong    cflag;  
} PMDWORDREGS
```

### Prototype In

pmapi.h

### Description

Structure describing the 32-bit extended x86 CPU registers

### Members

<i>eax</i>	Value of the EAX register
<i>ebx</i>	Value of the EBX register
<i>ecx</i>	Value of the ECX register
<i>edx</i>	Value of the EDX register
<i>esi</i>	Value of the ESI register
<i>edi</i>	Value of the EDI register
<i>cflag</i>	Value of the carry flag

## PMEnableWriteCombineErrors

### Declaration

```
typedef enum {
    PM_MTRR_ERR_OK = 0,
    PM_MTRR_ERR_NOT_SUPPORTED = -1,
    PM_MTRR_ERR_PARAMS = -2,
    PM_MTRR_ERR_NOT_4KB_ALIGNED = -3,
    PM_MTRR_ERR_BELOW_1MB = -4,
    PM_MTRR_ERR_NOT_ALIGNED = -5,
    PM_MTRR_ERR_OVERLAP = -6,
    PM_MTRR_ERR_TYPE_MISMATCH = -7,
    PM_MTRR_ERR_NONE_FREE = -8,
    PM_MTRR_ERR_NOWRCOMB = -9,
    PM_MTRR_ERR_NO_OS_SUPPORT = -10
} PMEnableWriteCombineErrors
```

### Prototype In

pmapi.h

### Description

Error codes returned by *PM\_enableWriteCombine*. Error code names are self explanatory.

## PMEnableWriteCombineFlags

### Declaration

```
typedef enum {
    PM_MTRR_UNCACHABLE    = 0,
    PM_MTRR_WRCOMB        = 1,
    PM_MTRR_WRTHROUGH     = 4,
    PM_MTRR_WRPROT        = 5,
    PM_MTRR_WRBACK        = 6,
    PM_MTRR_MAX           = 6,
} PMEnableWriteCombineFlags
```

### Prototype In

pmapi.h

### Description

Flags passed to *PM\_enableWriteCombine*

### Members

<i>PM_MTRR_UNCACHABLE</i>	Make region uncacheable
<i>PM_MTRR_WRCOMB</i>	Make region write combineable
<i>PM_MTRR_WRTHROUGH</i>	Make region write through cached
<i>PM_MTRR_WRPROT</i>	Make region write protected
<i>PM_MTRR_WRBACK</i>	Make region write back cached
<i>PM_MTRR_MAX</i>	Maximum value allowed



## PMFileFlagsType

### Declaration

```
typedef enum {  
    PM_FILE_NORMAL          = 0x00000000,  
    PM_FILE_READONLY        = 0x00000001,  
    PM_FILE_DIRECTORY       = 0x00000002,  
    PM_FILE_ARCHIVE         = 0x00000004,  
    PM_FILE_HIDDEN          = 0x00000008,  
    PM_FILE_SYSTEM          = 0x00000010  
} PMFileFlagsType
```

### Prototype In

pmapi.h

### Description

Flags stored in the *PM\_findData* structure, and also values passed to *PM\_setFileAttr* to change the file attributes. Flag names are self explanatory.

## PMREGS

### Declaration

```
typedef union {  
    PMDWORDREGS e;  
    PMWORDREGS x;  
    PMBYTEREGS h;  
} PMREGS
```

### Prototype In

pmapi.h

### Description

Structure describing all the x86 CPU registers

### Members

<i>e</i>	Member to access registers as 32-bit values
<i>x</i>	Member to access registers as 16-bit values
<i>h</i>	Member to access registers as 8-bit values

## PMSREGS

### Declaration

```
typedef struct {  
    ushort  es;  
    ushort  cs;  
    ushort  ss;  
    ushort  ds;  
    ushort  fs;  
    ushort  gs;  
} PMSREGS
```

### Prototype In

pmapi.h

### Description

Structure describing all the x86 segment registers

### Members

<i>es</i>	ES segment register
<i>cs</i>	CS segment register
<i>ss</i>	SS segment register
<i>ds</i>	DS segment register
<i>fs</i>	FS segment register
<i>gs</i>	GS segment register

## PMSplitPathFlags

### Declaration

```
typedef enum {  
    PM_HAS_WILDCARDS      = 0x01,  
    PM_HAS_EXTENSION      = 0x02,  
    PM_HAS_FILENAME       = 0x04,  
    PM_HAS_DIRECTORY      = 0x08,  
    PM_HAS_DRIVE          = 0x10  
} PMSplitPathFlags
```

### Prototype In

pmapi.h

### Description

Flags returned by the *PM\_splitpath* function. Flag names are self explanatory.

## PMWORDREGS

### Declaration

```
typedef struct {  
    ushort  ax,ax_hi;  
    ushort  bx,bx_hi;  
    ushort  cx,cx_hi;  
    ushort  dx,dx_hi;  
    ushort  si,si_hi;  
    ushort  di,di_hi;  
    ushort  cflag,cflag_hi;  
} PMWORDREGS
```

### Prototype In

pmapi.h

### Description

Structure describing the 16-bit x86 CPU registers

### Members

<i>ax</i>	Value of the AX register
<i>bx</i>	Value of the BX register
<i>cx</i>	Value of the CX register
<i>dx</i>	Value of the DX register
<i>si</i>	Value of the SI register
<i>di</i>	Value of the DI register
<i>cflag</i>	Value of the carry flag

## PM\_HWND

**Declaration**

```
typedef void          *PM_HWND
```

**Prototype In**

pmapi.h

**Description**

Fundamental type definition for a window handle. Note that the portable version of this define is simply to make it a void pointer, but internally it will be represented as a pointer to an internal operating system window handle.

## PM\_IRQHandle

**Declaration**

```
typedef void *PM_IRQHandle
```

**Prototype In**

pmapi.h

**Description**

Type definition for a Hardware IRQ handle used to save and restore the hardware IRQ handler.

## PM\_MODULE

**Declaration**

```
typedef void          *PM_MODULE
```

**Prototype In**

pmapi.h

**Description**

Fundamental type definition for a module handle. Note that the portable version of this define is simply to make it a void pointer, but internally it will be represented as a pointer to an internal operating system module handle.



## PM\_agpMemoryType

### Declaration

```
typedef enum {
    PM_agpUncached,
    PM_agpWriteCombine,
    PM_agpIntelDCACHE
} PM_agpMemoryType
```

### Prototype In

pmapi.h

### Description

This enumeration defines the type values passed to the *PM\_agpReservePhysical* function, to define how the physical memory mapping should be handled.

The *PM\_agpUncached* type indicates that the memory should be allocated as uncached memory.

The *PM\_agpWriteCombine* type indicates that write combining should be enabled for physical memory mapping. This is used for framebuffer write combining and speeds up direct framebuffer writes to the memory.

The *PM\_agpIntelDCACHE* type indicates that memory should come from the Intel i81x Display Cache (or DCACHE) memory pool. This flag is specific to the Intel i810 and i815 controllers, and should not be passed for any other controller type.

### Members

<i>PM_agpUncached</i>	Indicates that the memory should be uncached
<i>PM_agpWriteCombine</i>	Indicates that the memory should be write combined
<i>PM_agpIntelDCACHE</i>	Indicates that the memory should come from DCACHE pool

## PM\_enumWriteCombine\_t

### Declaration

```
typedef void (PMAPI PM_enumWriteCombine_t) (  
    ulong base,  
    ulong length,  
    uint type)
```

### Prototype In

pmapi.h

### Description

Type definition for enum write combined callback function

## **PM\_fatalCleanupHandler**

### **Declaration**

```
typedef void (PMAPI PM_fatalCleanupHandler) (void)
```

### **Prototype In**

pmapi.h

### **Description**

Type definition for the fatal cleanup handler

## PM\_findData

### Declaration

```
typedef struct {
    ulong    dwSize;
    ulong    attrib;
    ulong    sizeLo;
    ulong    sizeHi;
    char     name[PM_MAX_PATH];
} PM_findData
```

### Prototype In

pmapi.h

### Description

Structure for generic directory traversal and management. Also the same values are passed to *PM\_setFileAttr* to change the file attributes.

---

**Note:** *The dwSize member is intended for future compatibility, and should be set to the size of the structure as defined in this header file. Future drivers will be compatible with older software by examining this value.*

---

### Members

<i>dwSize</i>	Set to size of structure in bytes
<i>attrib</i>	Attributes for the file
<i>sizeLo</i>	Size of the file (low 32-bits)
<i>sizeHi</i>	Size of the file (high 32-bits)
<i>name</i>	Name of the file

## **PM\_intHandler**

### **Declaration**

```
typedef void (PMAPI PM_intHandler) (void)
```

### **Prototype In**

pmapi.h

### **Description**

Type definition for a C based interrupt handler

## PM\_irqHandler

**Declaration**

```
typedef ibool (PMAPIP PM_irqHandler) (  
    void *context)
```

**Prototype In**

pmapi.h

**Description**

Type definition for a C based hardware interrupt handler

## PM\_lockHandle

### Declaration

```
typedef struct {  
    ulong    h[3];  
} PM_lockHandle
```

### Prototype In

pmapi.h

### Description

Type definition for the lock handle used for locking linear memory

### Members

*h*            Internal lock handle details

## **PM\_physAddr**

### **Declaration**

```
typedef unsigned long PM_physAddr
```

### **Prototype In**

pmapi.h

### **Description**

Fundamental type definition for a physical memory address



## PM\_suspendAppCodesType

### Declaration

```
typedef enum {
    PM_SUSPEND_APP      = 0,
    PM_NO_SUSPEND_APP   = 1
} PM_suspendAppCodesType
```

### Prototype In

pmapi.h

### Description

Defines the return codes that the application can return from the suspend application callback registered with the PM library. The default value to be returned is PM\_SUSPEND\_APP and this will cause the application execution to be suspended until the application is re-activated again by the user. During this time the application will exist on the task bar under Windows 9x and Windows NT/2000/XP in minimised form.

PM\_NO\_SUSPEND\_APP can be used to tell the PM library to switch back to the Windows desktop, but not to suspend the applications execution. This must be used with care as the suspend application callback is then responsible for setting a flag in the application that will *stop* the application from doing any rendering directly to the framebuffer while the application is minimised on the task bar (since the application no longer owns the screen!). This return value is most useful for networked games that need to maintain network connectivity while the user has temporarily switched back to the Windows desktop. Hence you can ensure that your main loop continues to run, including networking and AI code, but no drawing occurs to the screen.

---

**Note:** *The PM library ensures that your application will never be switched away from outside of a message processing loop. Hence as long as you do not process messages inside your drawing loops, you will never lose the active focus (and your surfaces) while drawing, but only during event processing. The exception to this is if the user hits Ctrl-Alt-Del under Windows NT/2000/XP which will always cause a switch away from the application immediately and force the surfaces to be lost.*

---

### Members

<b>PM_SUSPEND_APP</b>	Suspend application execution until restored
<b>PM_NO_SUSPEND_APP</b>	Don't suspend execution, but allow switch

## PM\_suspendAppFlagsType

### Declaration

```
typedef enum {
    PM_DEACTIVATE    = 0x0001,
    PM_REACTIVATE    = 0x0002
} PM_suspendAppFlagsType
```

### Prototype In

pmapi.h

### Description

Defines the suspend application callback flags, passed to the suspend application callback registered with the PM library. This callback is called when the user presses one of the system key sequences indicating that they wish to change the active application. The PM library will catch these events and if you have registered a callback, will call the callback to save the state of the application so that it can be properly restored when the user switches back to your application.

---

**Note:** *Your application suspend callback may get called twice with the PM\_DEACTIVATE flag in order to test whether the switch should occur.*

**Note:** *When your callback is called with the PM\_DEACTIVATE flag, you cannot assume that you have access to the display memory surfaces as they may have been lost by the time your callback has been called.*

---

### Members

PM_DEACTIVATE	Application losing active focus
PM_REACTIVATE	Application regaining active focus

## PM\_suspendApp\_cb

**Declaration**

```
typedef int (PMAPI PM_suspendApp_cb) (  
    int flags)
```

**Prototype In**

pmapi.h

**Description**

Type definition for suspend application callback function

## PM\_time

### Declaration

```
typedef struct {  
    short    sec;  
    short    min;  
    short    hour;  
    short    day;  
    short    mon;  
    short    year;  
} PM_time
```

### Prototype In

pmapi.h

### Description

Structure passed to the *PM\_setFileTime* functions

### Members

<i>sec</i>	Seconds
<i>min</i>	Minutes
<i>hour</i>	Hour (0--23)
<i>day</i>	Day of month (1--31)
<i>mon</i>	Month (0--11)
<i>year</i>	Year (calendar year minus 1900)

## RMREGS

**Declaration**

```
typedef PMREGS  RMREGS
```

**Prototype In**

pmapi.h

**Description**

Same as *PMREGS*. Please see *PMREGS* for more information.

## RMSREGS

**Declaration**

```
typedef PMSREGS RMSREGS
```

**Prototype In**

pmapi.h

**Description**

Same as *PMSREGS*. Please see *PMSREGS* for more information.

## `__codePtr`

### **Declaration**

```
typedef void (*__codePtr) ()
```

### **Prototype In**

pmapi.h

### **Description**

Type definition for a generic code pointer

## codepage\_entry\_t

### Declaration

```
typedef struct {  
    uchar    scanCode;  
    uchar    asciiCode;  
} codepage_entry_t
```

### Prototype In

event.h

### Description

Structure describing an entry in the code page table. A table of translation codes for scan codes to ASCII codes is provided in this table to be used by the keyboard event libraries. On some OS'es the keyboard translation is handled by the OS, but for DOS and embedded systems you must register a different code page translation table if you want to support keyboards other than the US English keyboard (the default).

---

**Note:** *Entries in code page tables **must** be in ascending order for the scan codes as we do a binary search on the tables for the ASCII code equivalents.*

---

### Members

<i>scanCode</i>	Scan code to translate (really the virtual scan code).
<i>asciiCode</i>	ASCII code for this scan code.



## codepage\_t

### Declaration

```
typedef struct {
    char          name[20];
    codepage_entry_t *normal;
    int           normalLen;
    codepage_entry_t *caps;
    int           capsLen;
    codepage_entry_t *shift;
    int           shiftLen;
    codepage_entry_t *shiftCaps;
    int           shiftCapsLen;
    codepage_entry_t *ctrl;
    int           ctrlLen;
    codepage_entry_t *numPad;
    int           numPadLen;
} codepage_t
```

### Prototype In

event.h

### Description

Structure describing a complete code page translation table. The table contains translation tables for normal keys, shifted keys and ctrl keys. The Ctrl key always has precedence over the shift table, and the shift table is used when the shift key is down or the CAPSLOCK key is down.

## event\_t

### Declaration

```
typedef struct {
    unsigned long    which;
    unsigned long    what;
    unsigned long    when;
    int              where_x;
    int              where_y;
    int              relative_x;
    int              relative_y;
    unsigned long    message;
    unsigned long    modifiers;
    int              next;
    int              prev;
} event_t
```

### Prototype In

event.h

### Description

Structure describing the information contained in an event extracted from the event queue.

### Members

<i>which</i>	Window identifier for message for use by high level window manager code (i.e. MegaVision GUI or Windows API).
<i>what</i>	Type of event that occurred. Will be one of the values defined by the <i>EVT_eventType</i> enumeration.
<i>when</i>	Time that the event occurred in milliseconds since startup
<i>where_x</i>	X coordinate of the mouse cursor location at the time of the event (in screen coordinates). For joystick events this represents the position of the first joystick X axis.
<i>where_y</i>	Y coordinate of the mouse cursor location at the time of the event (in screen coordinates). For joystick events this represents the position of the first joystick Y axis.
<i>relative_x</i>	Relative movement of the mouse cursor in the X direction (in units of mickeys, or 1/200th of an inch). For joystick events this represents the position of the second joystick X axis.
<i>relative_y</i>	Relative movement of the mouse cursor in the Y direction (in units of mickeys, or 1/200th of an inch). For joystick events this represents the position of the second joystick Y axis.
<i>message</i>	Event specific message for the event. For use events this can be any user specific information. For keyboard events this contains the ASCII code in bits 0-7, the keyboard scan code in bits 8-15 and the character repeat count in bits 16-30. You can use the <i>EVT_asciiCode</i> , <i>EVT_scanCode</i> and

	<i>EVT_repeatCount</i> macros to extract this information from the message field. For mouse events this contains information about which button was pressed, and will be a combination of the flags defined by the <i>EVT_eventMouseMaskType</i> enumeration. For joystick events, this contains information about which buttons were pressed, and will be a combination of the flags defined by the <i>EVT_eventJoyMaskType</i> enumeration.
<i>modifiers</i>	Contains additional information about the state of the keyboard shift modifiers (Ctrl, Alt and Shift keys) when the event occurred. For mouse events it will also contain the state of the mouse buttons. Will be a combination of the values defined by the <i>EVT_eventModMaskType</i> enumeration.
<i>next</i>	Internal use; do not use.
<i>prev</i>	Internal use; do not use.

—  
\_\_codePtr, 822

## A

AlignLinearBuffer. *See*  
GA\_initFuncs::AlignLinearBuffer  
AllocBuffer. *See* GA\_bufferFuncs::AllocBuffer  
AllocVideoBuffer. *See*  
GA\_videoFuncs::AllocVideoBuffer

## B

BeginAccess. *See* GA\_cursorFuncs::BeginAccess  
BitBlt. *See* GA\_2DRenderFuncs::BitBlt  
BitBltBM. *See* GA\_2DRenderFuncs::BitBltBM  
BitBltBuf. *See* GA\_bufferFuncs::BitBltBuf  
BitBltColorPatt. *See*  
GA\_2DRenderFuncs::BitBltColorPatt  
BitBltColorPattBM. *See*  
GA\_2DRenderFuncs::BitBltColorPattBM  
BitBltColorPattBuf. *See*  
GA\_bufferFuncs::BitBltColorPattBuf  
BitBltColorPattLin. *See*  
GA\_2DRenderFuncs::BitBltColorPattLin  
BitBltColorPattSys. *See*  
GA\_2DRenderFuncs::BitBltColorPattSys  
BitBltFx. *See* GA\_2DRenderFuncs::BitBltFx  
BitBltFxBM. *See* GA\_2DRenderFuncs::BitBltFxBM  
BitBltFxBuf. *See* GA\_bufferFuncs::BitBltFxBuf  
BitBltFxFxLin. *See* GA\_2DRenderFuncs::BitBltFxFxLin  
BitBltFxFxSys. *See* GA\_2DRenderFuncs::BitBltFxFxSys  
BitBltFxFxTest. *See* GA\_2DRenderFuncs::BitBltFxFxTest  
BitBltLin. *See* GA\_2DRenderFuncs::BitBltLin  
BitBltPatt. *See* GA\_2DRenderFuncs::BitBltPatt  
BitBltPattBM. *See*  
GA\_2DRenderFuncs::BitBltPattBM  
BitBltPattBuf. *See* GA\_bufferFuncs::BitBltPattBuf  
BitBltPattLin. *See* GA\_2DRenderFuncs::BitBltPattLin  
BitBltPattSys. *See*  
GA\_2DRenderFuncs::BitBltPattSys  
BitBltPlaneMasked. *See*  
GA\_2DRenderFuncs::BitBltPlaneMasked  
BitBltPlaneMaskedBM. *See*  
GA\_2DRenderFuncs::BitBltPlaneMaskedBM  
BitBltPlaneMaskedBuf. *See*  
GA\_bufferFuncs::BitBltPlaneMaskedBuf  
BitBltPlaneMaskedLin. *See*  
GA\_2DRenderFuncs::BitBltPlaneMaskedLin  
BitBltPlaneMaskedSys. *See*  
GA\_2DRenderFuncs::BitBltPlaneMaskedSys

BitBltSys. *See* GA\_2DRenderFuncs::BitBltSys  
BuildTranslateVector. *See*  
GA\_2DStateFuncs::BuildTranslateVector

## C

ClearRegion. *See* GA\_regionFuncs::ClearRegion  
ClipEllipse. *See* GA\_2DRenderFuncs::ClipEllipse  
ClipMonoImageLSBBM. *See*  
GA\_2DRenderFuncs::ClipMonoImageLSBBM  
ClipMonoImageLSBLin. *See*  
GA\_2DRenderFuncs::ClipMonoImageLSBLin  
ClipMonoImageLSBSys. *See*  
GA\_2DRenderFuncs::ClipMonoImageLSBSys  
ClipMonoImageMSBBM. *See*  
GA\_2DRenderFuncs::ClipMonoImageMSBBM  
ClipMonoImageMSBLin. *See*  
GA\_2DRenderFuncs::ClipMonoImageMSBLin  
ClipMonoImageMSBSys. *See*  
GA\_2DRenderFuncs::ClipMonoImageMSBSys  
codepage\_entry\_t, 823  
codepage\_t, 824  
CopyIntoRegion. *See*  
GA\_regionFuncs::CopyIntoRegion  
CopyRegion. *See* GA\_regionFuncs::CopyRegion  
CPU\_getProcessorName, 589, 590, 591, 592, 593,  
594, 595, 596  
CPU\_getProcessorSpeed, 590, 591, 592, 593, 594,  
595, 596  
CPU\_getProcessorSpeedInHZ, 591  
CPU\_getProcessorType, 589, 590, 591, 592, 593, 594,  
595, 596, 761  
CPU\_have3DNow, 593, 594, 595, 596  
CPU\_haveMMX, 589, 590, 591, 592, 593, 594, 595,  
596  
CPU\_haveRDTSC, 595  
CPU\_haveSSE, 593, 594, 596  
CPU\_largeInteger, 760  
CPU\_processorType, 761  
CreateClipper. *See* GA\_clipperFuncs::CreateClipper

## D

DDC\_ChannelsType, 158, 329, 331, 332, 333, 334,  
335  
DDC\_DPMSStatesType, 159, 324  
DDC\_errCode, 45, 46, 121, 161  
DDC\_init, 45  
DDC\_initExt, 45, 46, 47, 48, 49, 161  
DDC\_readEDID, 46, 47, 48, 49, 124  
DDC\_SCIFlagsType, 160, 330  
DDC\_writeEDID, 48

- DestroyClipper. *See*
    - GA\_clipperFuncs::DestroyClipper
  - DiffRegion. *See* GA\_regionFuncs::DiffRegion
  - DiffRegionRect. *See*
    - GA\_regionFuncs::DiffRegionRect
  - DisableDirectAccess. *See*
    - GA\_2DStateFuncs::DisableDirectAccess
  - DPMSdetect. *See* GA\_DPMSFuncs::DPMSdetect
  - DPMSsetState. *See* GA\_DPMSFuncs::DPMSsetState
  - DrawBresenhamLine. *See*
    - GA\_2DRenderFuncs::DrawBresenhamLine
  - DrawBresenhamStippleLine. *See*
    - GA\_2DRenderFuncs::DrawBresenhamStippleLine
  - DrawBresenhamStyleLine. *See*
    - GA\_2DRenderFuncs::DrawBresenhamStyleLine
  - DrawClippedBresenhamLine. *See*
    - GA\_2DRenderFuncs::DrawClippedBresenhamLine
  - DrawClippedBresenhamStippleLine. *See*
    - GA\_2DRenderFuncs::DrawClippedBresenhamStippleLine
  - DrawClippedBresenhamStyleLine. *See*
    - GA\_2DRenderFuncs::DrawClippedBresenhamStyleLine
  - DrawClippedLineInt. *See*
    - GA\_2DRenderFuncs::DrawClippedLineInt
  - DrawClippedStippleLineInt. *See*
    - GA\_2DRenderFuncs::DrawClippedStippleLineInt
  - DrawClippedStyleLineInt. *See*
    - GA\_2DRenderFuncs::DrawClippedStyleLineInt
  - DrawColorPattEllipseList. *See*
    - GA\_2DRenderFuncs::DrawColorPattEllipseList
  - DrawColorPattFatEllipseList. *See*
    - GA\_2DRenderFuncs::DrawColorPattFatEllipseList
  - DrawColorPattRect. *See*
    - GA\_2DRenderFuncs::DrawColorPattRect
  - DrawColorPattScanList. *See*
    - GA\_2DRenderFuncs::DrawColorPattScanList
  - DrawColorPattTrap. *See*
    - GA\_2DRenderFuncs::DrawColorPattTrap
  - DrawEllipse. *See* GA\_2DRenderFuncs::DrawEllipse
  - DrawEllipseList. *See*
    - GA\_2DRenderFuncs::DrawEllipseList
  - DrawFatEllipseList. *See*
    - GA\_2DRenderFuncs::DrawFatEllipseList
  - DrawLineInt. *See* GA\_2DRenderFuncs::DrawLineInt
  - DrawPattEllipseList. *See*
    - GA\_2DRenderFuncs::DrawPattEllipseList
  - DrawPattFatEllipseList. *See*
    - GA\_2DRenderFuncs::DrawPattFatEllipseList
  - DrawPattRect. *See*
    - GA\_2DRenderFuncs::DrawPattRect
  - DrawPattScanList. *See*
    - GA\_2DRenderFuncs::DrawPattScanList
  - DrawPattTrap. *See*
    - GA\_2DRenderFuncs::DrawPattTrap
  - DrawRect. *See* GA\_2DRenderFuncs::DrawRect
  - DrawRectBuf. *See* GA\_bufferFuncs::DrawRectBuf
  - DrawRectExt. *See*
    - GA\_2DRenderFuncs::DrawRectExt
  - DrawRectExtSW. *See*
    - REF2D\_driver::DrawRectExtSW
  - DrawRectLin. *See*
    - GA\_2DRenderFuncs::DrawRectLin
  - DrawScanList. *See*
    - GA\_2DRenderFuncs::DrawScanList
  - DrawStippleLineInt. *See*
    - GA\_2DRenderFuncs::DrawStippleLineInt
  - DrawStyleLineInt. *See*
    - GA\_2DRenderFuncs::DrawStyleLineInt
  - DrawTrap. *See* GA\_2DRenderFuncs::DrawTrap
  - DstTransBlit. *See* GA\_2DRenderFuncs::DstTransBlit
  - DstTransBlitBM. *See*
    - GA\_2DRenderFuncs::DstTransBlitBM
  - DstTransBlitBuf. *See*
    - GA\_bufferFuncs::DstTransBlitBuf
  - DstTransBlitLin. *See*
    - GA\_2DRenderFuncs::DstTransBlitLin
  - DstTransBlitSys. *See*
    - GA\_2DRenderFuncs::DstTransBlitSys
- ## E
- EDID\_detailedTiming, 162
  - EDID\_displayTypes, 163, 167
  - EDID\_flags, 164, 167
  - EDID\_maxResCodes, 165, 167
  - EDID\_parse, 47, 49
  - EDID\_record, 163, 164, 165, 166, 168
  - EDID\_signalLevels, 167, 168
  - EDID\_standardTiming, 169, 170
  - EDID\_timingTypes, 169, 170
  - EnableDirectAccess. *See*
    - GA\_2DStateFuncs::EnableDirectAccess
  - EnableStereoMode. *See*
    - GA\_driverFuncs::EnableStereoMode
  - EndAccess. *See* GA\_cursorFuncs::EndAccess
  - EndVideoFrame. *See*
    - GA\_videoFuncs::EndVideoFrame
  - event\_t, 598, 603, 769, 771, 825
  - EVT\_allowLEDS, 597
  - EVT\_asciiCode, 598, 613, 614, 765, 825
  - EVT\_asciiCodesType, 763
  - EVT\_eventJoyAxisType, 766
  - EVT\_eventJoyMaskType, 767, 825
  - EVT\_eventMaskType, 768, 771
  - EVT\_eventModMaskType, 769, 826
  - EVT\_eventMouseMaskType, 770, 825
  - EVT\_eventType, 603, 610, 771, 825
  - EVT\_flush, 599, 603, 604, 610, 612
  - EVT\_getCodePage, 600, 615
  - EVT\_getHeartBeatCallback, 601, 616
  - EVT\_getMousePos, 602, 617
  - EVT\_getNext, 599, 601, 603, 604, 610, 611, 612, 616, 618
  - EVT\_halt, 599, 603, 604, 610, 612
  - EVT\_isKeyDown, 605
  - EVT\_joyIsPresent, 606, 608, 609, 611
  - EVT\_joySetCenter, 606, 607, 608, 611
  - EVT\_joySetLowerRight, 606, 607, 608, 609, 611
  - EVT\_joySetUpperLeft, 607, 608, 609, 611
  - EVT\_masksType, 772
  - EVT\_peekNext, 599, 601, 603, 604, 610, 611, 612, 616, 618

EVT\_pollJoystick, 611  
 EVT\_post, 612  
 EVT\_repeatCount, 598, 613, 614, 825  
 EVT\_scanCode, 598, 614, 774, 825  
 EVT\_scanCodesType, 614, 773  
 EVT\_setCodePage, 600, 615  
 EVT\_setHeartBeatCallback, 601, 616  
 EVT\_setMousePos, 602, 617  
 EVT\_setUserEventFilter, 618

## F

FlipToBuffer. *See* GA\_bufferFuncs::FlipToBuffer  
 FlipToStereoBuffer. *See*  
     GA\_bufferFuncs::FlipToStereoBuffer  
 ForceSoftwareOnly. *See*  
     REF2D\_driver::ForceSoftwareOnly  
 FreeBuffer. *See* GA\_bufferFuncs::FreeBuffer  
 FreeRegion. *See* GA\_regionFuncs::FreeRegion  
 FreeVideoBuffer. *See*  
     GA\_videoFuncs::FreeVideoBuffer

## G

GA\_2DRenderFuncs, 171, 287, 288, 299, 300, 301, 302, 377, 446  
 BitBlt, 172, 174, 175, 180, 182, 184, 186, 188, 189, 190, 194, 196, 197, 198, 200, 201, 248, 250, 252, 254, 255, 256, 266, 268, 270, 272, 274, 276, 278, 280  
 BitBltBM, 172, 173, 189, 201  
 BitBltColorPatt, 175, 176, 177, 178  
 BitBltColorPattBM, 175, 176, 177, 178  
 BitBltColorPattLin, 175, 176, 177  
 BitBltColorPattSys, 175, 176, 177, 178  
 BitBltFx, 172, 174, 179, 182, 184, 186, 187, 189, 201, 248, 250, 252, 254, 266, 268, 270, 272, 278  
 BitBltFxBM, 180, 181, 182, 184, 186, 187, 250, 268  
 BitBltFxLin, 180, 182, 183, 186, 187, 252, 270  
 BitBltFxSys, 180, 182, 184, 185, 187, 253, 271  
 BitBltFxTest, 179, 180, 182, 184, 186, 187, 362, 363  
 BitBltLin, 172, 174, 177, 188, 192  
 BitBltPatt, 190, 191, 192, 193  
 BitBltPattBM, 190, 191, 192, 193  
 BitBltPattLin, 190, 191, 192  
 BitBltPattSys, 190, 191, 192, 193  
 BitBltPlaneMasked, 194, 196, 198, 200  
 BitBltPlaneMaskedBM, 194, 195, 198, 200  
 BitBltPlaneMaskedLin, 194, 196, 197, 200  
 BitBltPlaneMaskedSys, 194, 196, 198, 199  
 BitBltSys, 172, 173, 174, 176, 178, 189, 191, 193, 195, 201  
 ClipEllipse, 202, 230  
 ClipMonoImageLSBBM, 203, 204, 205, 206, 207, 208, 258, 259, 260, 261, 262, 264  
 ClipMonoImageLSBLin, 204, 205, 206, 207, 208, 258, 259, 260, 261, 262, 264  
 ClipMonoImageLSBSys, 203, 204, 205, 258, 259, 260, 261, 262, 264

ClipMonoImageMSBBM, 203, 204, 205, 206, 207, 208, 258, 259, 260, 261, 262, 264  
 ClipMonoImageMSBLin, 203, 204, 205, 207, 208, 258, 259, 260, 261, 262, 264  
 ClipMonoImageMSBSys, 203, 204, 205, 206, 207, 208, 258, 259, 260, 261, 262, 264  
 DrawBresenhamLine, 209, 234  
 DrawBresenhamStippleLine, 211, 245, 294, 295  
 DrawBresenhamStyleLine, 213, 246, 296  
 DrawClippedBresenhamLine, 215, 221  
 DrawClippedBresenhamStippleLine, 217, 222  
 DrawClippedBresenhamStyleLine, 219, 223  
 DrawClippedLineInt, 216, 218, 220, 221, 222, 223, 234  
 DrawClippedStippleLineInt, 216, 218, 221, 222  
 DrawClippedStyleLineInt, 220, 223  
 DrawColorPattEllipseList, 224, 226, 231, 233, 235, 237  
 DrawColorPattFatEllipseList, 224, 225, 231, 233, 235, 237  
 DrawColorPattRect, 227, 238, 241, 287, 299, 301  
 DrawColorPattScanList, 228, 239, 244, 287, 299, 301  
 DrawColorPattTrap, 229, 240, 247, 287, 299, 301  
 DrawEllipse, 202, 230  
 DrawEllipseList, 224, 226, 231, 233, 235, 237  
 DrawFatEllipseList, 224, 226, 231, 232, 235, 237  
 DrawLineInt, 209, 210, 212, 214, 215, 234, 245, 246  
 DrawPattEllipseList, 224, 226, 231, 233, 235, 237  
 DrawPattFatEllipseList, 224, 226, 231, 233, 235, 236  
 DrawPattRect, 227, 238, 241, 288, 300, 302  
 DrawPattScanList, 239, 244, 288, 300, 302  
 DrawPattTrap, 240, 247, 288, 300, 302  
 DrawRect, 227, 238, 241, 242, 243, 366  
 DrawRectExt, 241, 242, 243  
 DrawRectLin, 241, 242, 243  
 DrawScanList, 228, 229, 239, 240, 244, 247  
 DrawStippleLineInt, 210, 211, 212, 213, 217, 219, 234, 245, 294, 295  
 DrawStyleLineInt, 214, 234, 246, 296  
 DrawTrap, 229, 240, 247  
 DstTransBlt, 172, 174, 175, 176, 177, 178, 180, 182, 184, 186, 189, 190, 191, 192, 193, 201, 248, 250, 252, 254, 266, 268, 270, 272, 274, 276, 278, 280  
 DstTransBltBM, 248, 249, 252, 254  
 DstTransBltLin, 248, 250, 251, 254  
 DstTransBltSys, 248, 250, 252, 253  
 GetBitmapBM, 255, 256  
 GetBitmapSys, 255, 256  
 GetPixel, 257, 265  
 PutMonoImageLSBBM, 203, 204, 205, 206, 207, 208, 258, 259, 260, 261, 262, 264  
 PutMonoImageLSBLin, 203, 204, 205, 206, 207, 208, 259, 260, 261, 262, 264  
 PutMonoImageLSBSys, 203, 204, 205, 206, 207, 208, 258, 259, 260  
 PutMonoImageMSBBM, 203, 204, 205, 206, 207, 208, 258, 259, 260, 261, 262, 264

- PutMonoImageMSBLine, 203, 204, 205, 206, 207, 208, 258, 259, 260, 262, 264
- PutMonoImageMSBSys, 203, 204, 205, 206, 207, 208, 258, 259, 260, 261, 262, 263, 264
- PutPixel, 257, 265
- SrcTransBlt, 172, 174, 175, 176, 177, 178, 180, 182, 184, 186, 189, 190, 191, 192, 193, 201, 266, 268, 270, 272, 274, 276, 278, 280
- SrcTransBltBM, 266, 267, 270, 272
- SrcTransBltLine, 266, 268, 269, 272
- SrcTransBltSys, 266, 268, 270, 271
- StretchBlt, 273, 276, 278, 280
- StretchBltBM, 274, 275, 278, 280
- StretchBltLine, 274, 276, 277, 280
- StretchBltSys, 274, 276, 278, 279
- UpdateScreen, 281
- GA\_2DStateFuncs, 282, 446
  - BuildTranslateVector, 283
  - DisableDirectAccess, 284, 285, 286, 303
  - EnableDirectAccess, 284, 285, 286, 303
  - IsIdle, 286, 303
  - Set8x8ColorPattern, 227, 228, 229, 287, 288, 290, 293, 297, 299, 300, 301, 302
  - Set8x8MonoPattern, 238, 239, 240, 287, 288, 290, 293, 297, 299, 300, 301, 302
  - SetAlphaValue, 289, 291
  - SetBackColor, 290, 293, 297
  - SetBlendFunc, 289, 291
  - SetDrawBuffer, 292, 585
  - SetForeColor, 290, 293, 297
  - SetLineStipple, 212, 218, 222, 245, 294, 295, 296
  - SetLineStippleCount, 212, 218, 222, 245, 294, 295, 296
  - SetLineStyle, 214, 220, 223, 246, 294, 295, 296
  - SetMix, 290, 293, 297
  - SetPlaneMask, 298
  - Use8x8ColorPattern, 227, 228, 229, 287, 288, 299, 300, 301, 302
  - Use8x8MonoPattern, 238, 239, 240, 287, 288, 299, 300, 301, 302
  - Use8x8TransColorPattern, 301
  - Use8x8TransMonoPattern, 300, 302
  - WaitTillIdle, 284, 285, 286, 303
- GA\_AccelFlagsType, 304
- GA\_addMode, 50, 51, 53, 55, 58
- GA\_addRefresh, 50, 51, 53
- GA\_AttributeExtFlagsType, 305, 503
- GA\_AttributeFlagsType, 306, 421, 501
- GA\_BitBltFxFlagsType, 310, 354
- GA\_blendFuncType, 291, 350, 354
- GA\_bltFx, 179, 181, 182, 183, 185, 187, 310, 312, 353, 362, 503
- GA\_BresenhamLineFlagsType, 209, 211, 213, 215, 217, 219, 313
- GA\_buf, 355, 376
- GA\_buffer, 292, 357
- GA\_BufferFlagsType, 314, 355, 359, 375
- GA\_bufferFuncs, 292, 358, 447
  - AllocBuffer, 359, 370, 373, 374, 375, 377
  - BitBltBuf, 359, 360, 361, 363, 364, 365, 366, 367, 371, 375, 378, 379, 381, 383, 384
  - BitBltColorPattBuf, 360, 361, 363, 364, 365, 367, 379, 381
  - BitBltFxBuf, 360, 361, 362, 364, 365, 367, 379, 381
  - BitBltPattBuf, 360, 361, 363, 364, 365, 367, 379, 381
  - BitBltPlaneMaskedBuf, 360, 361, 364, 365, 367, 379, 381
  - DrawRectBuf, 360, 361, 363, 364, 365, 366, 367, 379, 381
  - DstTransBltBuf, 360, 361, 363, 364, 365, 367, 379, 381
  - FlipToBuffer, 359, 368, 369, 372, 373, 374, 375, 377, 385, 428, 436, 437, 438
  - FlipToStereoBuffer, 368, 369, 372, 385, 426, 443
  - FreeBuffer, 359, 370
  - GetClipper, 371, 378
  - GetFlippableBuffer, 359, 373, 374
  - GetFlipStatus, 368, 369, 372, 385, 426, 428
  - GetPrimaryBuffer, 359, 373, 374, 375, 377
  - InitBuffers, 375
  - LockBuffer, 284, 285, 359, 376, 382
  - SetActiveBuffer, 359, 368, 369, 372, 373, 374, 375, 377, 385, 426, 428, 437, 438, 443
  - SetClipper, 378
  - SrcTransBltBuf, 360, 361, 363, 364, 365, 367, 379, 381
  - StretchBltBuf, 380
  - UnlockBuffer, 284, 285, 376, 382
  - UpdateCache, 383, 384
  - UpdateFromCache, 383, 384
  - WaitTillFlipped, 368, 369, 372, 385
- GA\_busType, 386, 424
- GA\_certifyChipInfo, 320, 387
- GA\_CertifyFlagsType, 320
- GA\_certifyInfo, 387, 388
- GA\_clipper, 389
- GA\_clipperFuncs, 390, 447
  - CreateClipper, 391, 392, 393, 394
  - DestroyClipper, 391, 392, 393, 394
  - GetClipList, 391, 392, 393, 394
  - IsClipListChanged, 391, 392, 393, 394
- GA\_color, 395
- GA\_colorCursor, 396, 412
- GA\_colorCursor256, 397, 413
- GA\_colorCursorRGB, 398, 414
- GA\_colorCursorRGBA, 399, 415
- GA\_colorPattern, 400
- GA\_colorPattern\_1, 401
- GA\_colorPattern\_16, 402
- GA\_colorPattern\_24, 403
- GA\_colorPattern\_32, 404
- GA\_colorPattern\_4, 405
- GA\_colorPattern\_8, 406
- GA\_computeCRTCTimings, 52, 71
- GA\_configInfo, 407
- GA\_CRTCInfo, 317, 319
- GA\_CRTCInfoFlagsType, 317, 319
- GA\_cursorFuncs, 408, 446
  - BeginAccess, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419

- EndAccess, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419
- IsHardwareCursor, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419
- SetColorCursor, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419
- SetColorCursor256, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419
- SetColorCursorRGB, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419
- SetColorCursorRGBA, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419
- SetCursorPos, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419
- SetMonoCursor, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419
- SetMonoCursorColor, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419
- ShowCursor, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419
- GA\_delMode, 50, 51, 53, 55, 58
- GA\_detectPnPMonitor, 54, 72, 463, 481
- GA\_devCtx, 87, 305, 306, 308, 347, 386, 420, 460, 468, 476, 485, 504
- GA\_disableVBEMode, 53, 55, 58
- GA\_disjointRect, 56
- GA\_DPMSFuncs, 322, 446
  - DPMSdetect, 323, 324
  - DPMSsetState, 323, 324
- GA\_driverFuncs, 425, 446
  - EnableStereoMode, 426, 443
  - GetCurrentScanLine, 427
  - GetDisplayStartStatus, 426, 428, 436, 437, 438, 443
  - GetGammaCorrectData, 429, 430, 431, 432, 439, 440, 441, 442
  - GetGammaCorrectDataExt, 429, 430, 431, 432, 439, 440, 441, 442
  - GetPaletteData, 338, 429, 430, 431, 432, 439, 440, 441, 442
  - GetPaletteDataExt, 429, 430, 431, 432, 439, 440, 441, 442
  - GetVSyncWidth, 433, 444
  - IsVSync, 434, 445
  - SetBank, 435
  - SetDisplayStart, 428, 436, 438, 443
  - SetDisplayStartXY, 437, 438
  - SetGammaCorrectData, 429, 430, 431, 432, 439, 440, 441, 442
  - SetGammaCorrectDataExt, 429, 430, 431, 432, 439, 440, 441, 442
  - SetPaletteData, 341, 429, 430, 431, 432, 439, 440, 441, 442
  - SetPaletteDataExt, 429, 430, 431, 432, 439, 440, 441, 442
  - SetStereoDisplayStart, 426, 437, 438, 443
  - SetVSyncWidth, 433, 444
  - WaitVSync, 434, 445
- GA\_emptyRect, 56, 57
- GA\_enableVBEMode, 50, 55, 58
- GA\_enumerateDevices, 59
- GA\_equalRect, 56, 60
- GA\_errorMsg, 61, 113
- GA\_funcGroupsType, 87, 446, 493
- GA\_getCRTCTimings, 62, 453, 472, 479
- GA\_getCurrentRef2d, 63, 82
- GA\_getDaysLeft, 64
- GA\_getDisplaySerialNo, 65
- GA\_getDisplayUserName, 66
- GA\_getFakePCIID, 67
- GA\_getGlobalOptions, 68, 89, 93, 104, 449
- GA\_getInternalName, 69
- GA\_getLicensedDevices, 70
- GA\_getMaxRefreshRate, 52, 71
- GA\_getParsedEDID, 54, 72
- GA\_getSNAPConfigPath, 73
- GA\_globalOptions, 325, 448
- GA\_initFuncs, 446, 450
  - AlignLinearBuffer, 292, 451
  - GetActiveHead, 452, 464, 474
  - GetCertifyInfo, 454, 457
  - GetClosestPixelClock, 455, 477, 487
  - GetConfigInfo, 454, 457
  - GetCRTCTimings, 453, 472, 475, 479
  - GetCurrentRefreshRate, 453, 458, 459, 461, 468, 469, 472, 479
  - GetCurrentVideoModeInfo, 458, 459, 461, 468, 469
  - GetCustomVideoModeInfo, 456, 458, 459, 460, 461, 467, 468, 469, 476, 477, 487
  - GetCustomVideoModeInfoExt, 458, 459, 461, 468, 469
  - GetDisplayOutput, 462
  - GetMonitorInfo, 463, 481
  - GetNumberOfHeads, 452, 464, 474
  - GetOptions, 465, 482
  - GetUniqueFilename, 466
  - GetVideoMode, 459, 467
  - GetVideoModeInfo, 456, 459, 460, 461, 467, 468, 469, 474, 477, 480, 486, 487
  - GetVideoModeInfoExt, 458, 459, 461, 468, 469
  - PerformDisplaySwitch, 470, 471, 488, 581
  - PollForDisplaySwitch, 470, 471, 488, 581
  - SaveCRTCTimings, 472, 475
  - SaveRestoreState, 473
  - SetActiveHead, 452, 464, 474
  - SetCRTCTimings, 453, 472, 475, 479
  - SetCustomVideoMode, 456, 460, 476, 487
  - SetDisplayOutput, 477, 478, 487
  - SetGlobalRefresh, 453, 472, 479
  - SetModeProfile, 480
  - SetMonitorInfo, 463, 481
  - SetOptions, 465, 482
  - SetRef2dPointer, 483, 484
  - SetSoftwareRenderFuncs, 483, 484
  - SetVideoMode, 456, 458, 459, 460, 461, 467, 468, 469, 474, 476, 477, 485
  - SwitchPhysicalResolution, 470, 471, 488
- GA\_insetRect, 74, 84
- GA\_isLiteVersion, 75
- GA\_isOEMVersion, 76
- GA\_isSharedDriverLoaded, 77, 79
- GA\_isSimpleRegion, 78
- GA\_largeInteger, 489



- GA\_layout, 490
- GA\_LCDUseBIOSFlagsType, 325
- GA\_loadDriver, 59, 79, 80, 101, 105, 113, 116
- GA\_loaderFuncs, 491
  - InitDriver, 492, 494, 495
  - QueryFunctions, 492, 493, 494, 495
  - UnloadDriver, 492, 494, 495
- GA\_loadInGUI, 80
- GA\_loadModeProfile, 81, 94
- GA\_loadRef2d, 63, 82, 154
- GA\_loadRegionMgr, 83, 118
- GA\_MakeVisibleBufferFlagsType, 326, 368, 369
- GA\_mixCodesType, 172, 173, 188, 201, 242, 243, 248, 249, 251, 253, 255, 256, 266, 267, 269, 271, 273, 275, 277, 279, 297, 360, 367, 379, 380, 496
- GA\_mode, 498
- GA\_modeFlagsType, 476, 485, 486, 499
- GA\_modeInfo, 183, 184, 187, 189, 198, 243, 251, 252, 259, 262, 269, 270, 277, 278, 292, 305, 306, 308, 310, 312, 319, 342, 345, 357, 395, 418, 421, 426, 436, 443, 451, 459, 460, 461, 468, 469, 486, 500, 556
- GA\_modeProfile, 506
- GA\_monitor, 507, 508
- GA\_monitorFlagsType, 508
- GA\_monoCursor, 417, 509
- GA\_multiHeadType, 452, 474, 510
- GA\_offsetRect, 74, 84
- GA\_options, 510, 511
- GA\_OutputFlagsType, 327, 461, 469, 478
- GA\_palette, 418, 429, 431, 439, 441, 517
- GA\_paletteExt, 432, 442, 518
- GA\_pattern, 519
- GA\_pixelFormat, 520
- GA\_programMTRRegisters, 85
- GA\_ptInRect, 86
- GA\_queryFunctions, 87, 155, 171, 282, 322, 328, 337, 358, 390, 408, 425, 450, 494, 526, 555, 578, 582
- GA\_readGlobalOptions, 89
- GA\_recMode, 523
- GA\_rect, 524
- GA\_region, 525
- GA\_regionFuncs, 447, 526
  - ClearRegion, 527, 528, 529
  - CopyIntoRegion, 528, 529
  - CopyRegion, 527, 528, 529, 532
  - DiffRegion, 78, 530, 531, 533, 534, 535, 536, 537, 538, 540, 541, 542, 543, 544, 545
  - DiffRegionRect, 530, 531
  - FreeRegion, 527, 528, 529, 532, 535, 536
  - IsEmptyRegion, 533, 534
  - IsEqualRegion, 533, 534
  - NewRectRegion, 535, 536
  - NewRegion, 527, 528, 529, 532, 535, 536
  - OffsetRegion, 533, 534, 537
  - OptimizeRegion, 538
  - PtInRegion, 533, 534, 539
  - SectRegion, 78, 530, 531, 533, 534, 535, 536, 537, 538, 540, 541, 542, 543, 544, 545
  - SectRegionRect, 540, 541
  - TraverseRegion, 542
  - UnionRegion, 78, 530, 531, 533, 534, 535, 536, 537, 538, 540, 541, 542, 543, 544, 545
  - UnionRegionOfs, 543, 544, 545
  - UnionRegionRect, 543, 544, 545
- GA\_registerLicense, 90
- GA\_restoreCRTCTimings, 62, 91, 92, 102, 103, 453, 472, 479
- GA\_rop3CodesType, 175, 176, 177, 178, 190, 191, 192, 193, 361, 364, 546
- GA\_saveCRTCTimings, 62, 91, 92, 94, 96, 102, 103, 453, 472, 479
- GA\_saveGlobalOptions, 68, 89, 93, 104
- GA\_saveModeProfile, 81, 91, 92, 94, 96, 480
- GA\_saveMonitorInfo, 95, 463, 481
- GA\_saveOptions, 91, 92, 94, 96, 465, 482
- GA\_SCIFuncs, 328, 446
  - SClbegin, 329, 330, 331, 332, 333, 334, 335
  - SCIdetect, 329, 330, 331, 332, 333, 334, 335
  - SClend, 329, 330, 331, 332, 333, 334, 335
  - SClreadSCL, 329, 330, 331, 332, 333, 334, 335
  - SClreadSDA, 329, 330, 331, 332, 333, 334, 335
  - SClwriteSCL, 329, 330, 331, 332, 333, 334, 335
  - SClwriteSDA, 329, 330, 331, 332, 333, 334, 335
- GA\_sectRect, 56, 97, 98, 99, 100, 114, 115
- GA\_sectRectCoord, 97, 98, 99
- GA\_sectRectFast, 97, 99
- GA\_sectRectFastCoord, 98, 100
- GA\_segment, 551
- GA\_setActiveDevice, 79, 101
- GA\_setCRTCTimings, 102, 453, 472, 479
- GA\_setDefaultRefresh, 103, 453, 472, 479
- GA\_setGlobalOptions, 68, 89, 93, 104
- GA\_setMinimumDriverVersion, 105
- GA\_softStereoExit, 106, 108, 109
- GA\_softStereoGetFlipStatus, 107, 110, 111, 112
- GA\_softStereoInit, 106, 108
- GA\_softStereoOff, 109, 110
- GA\_softStereoOn, 108, 109, 110
- GA\_softStereoScheduleFlip, 107, 110, 111, 112
- GA\_softStereoWaitTillFlipped, 107, 111, 112
- GA\_span, 552
- GA\_status, 61, 93, 94, 95, 96, 113
- GA\_stipple, 553
- GA\_trap, 229, 240, 247, 554
- GA\_TVParams, 336
- GA\_unionRect, 56, 97, 98, 99, 100, 114
- GA\_unionRectCoord, 114, 115
- GA\_unloadDriver, 79, 82, 116, 117
- GA\_unloadRef2d, 63, 82, 117
- GA\_unloadRegionMgr, 83, 118
- GA\_useDoubleScan, 119
- GA\_VBEFuncs, 337, 447
  - GetPaletteData, 338, 339, 340, 341
  - Set8BitDAC, 338, 339, 340, 341
  - SetBytesPerLine, 338, 339, 340, 341
  - SetPaletteData, 338, 339, 340, 341
- GA\_VideoBufferFormatsType, 342, 556
- GA\_videoFuncs, 446, 555
  - AllocVideoBuffer, 556, 557, 558, 559, 560, 561
  - EndVideoFrame, 556, 557, 558, 559, 560, 561
  - FreeVideoBuffer, 556, 557, 558, 559, 560, 561
  - SetVideoColorKey, 556, 557, 558, 559, 560, 561

SetVideoOutput, 556, 557, 558, 559, 560, 561  
 StartVideoFrame, 556, 557, 558, 559, 560, 561  
 GA\_videoInf, 342, 345, 503, 562  
 GA\_VideoOutputFlagsType, 345, 560  
 GA\_WorkAroundsFlagsType, 347  
 GetActiveHead. *See* GA\_initFuncs::GetActiveHead  
 GetBitmapBM. *See*  
     GA\_2DRenderFuncs::GetBitmapBM  
 GetBitmapSys. *See*  
     GA\_2DRenderFuncs::GetBitmapSys  
 GetCertifyInfo. *See* GA\_initFuncs::GetCertifyInfo  
 GetClipList. *See* GA\_clipperFuncs::GetClipList  
 GetClipper. *See* GA\_bufferFuncs::GetClipper  
 GetClosestPixelClock. *See*  
     GA\_initFuncs::GetClosestPixelClock  
 GetConfigInfo. *See* GA\_initFuncs::GetConfigInfo  
 GetCRTCTimings. *See*  
     GA\_initFuncs::GetCRTCTimings  
 GetCurrentRefreshRate. *See*  
     GA\_initFuncs::GetCurrentRefreshRate  
 GetCurrentScanLine. *See*  
     GA\_driverFuncs::GetCurrentScanLine  
 GetCurrentVideoModeInfo. *See*  
     GA\_initFuncs::GetCurrentVideoModeInfo  
 GetCustomVideoModeInfo. *See*  
     GA\_initFuncs::GetCustomVideoModeInfo  
 GetCustomVideoModeInfoExt. *See*  
     GA\_initFuncs::GetCustomVideoModeInfoExt  
 GetDisplayOutput. *See*  
     GA\_initFuncs::GetDisplayOutput  
 GetDisplayStartStatus. *See*  
     GA\_driverFuncs::GetDisplayStartStatus  
 GetFlippableBuffer. *See*  
     GA\_bufferFuncs::GetFlippableBuffer  
 GetFlipStatus. *See* GA\_bufferFuncs::GetFlipStatus  
 GetGammaCorrectData. *See*  
     GA\_driverFuncs::GetGammaCorrectData  
 GetGammaCorrectDataExt. *See*  
     GA\_driverFuncs::GetGammaCorrectDataExt  
 GetMonitorInfo. *See* GA\_initFuncs::GetMonitorInfo  
 GetNumberOfHeads. *See*  
     GA\_initFuncs::GetNumberOfHeads  
 GetOptions. *See* GA\_initFuncs::GetOptions  
 GetPaletteData. *See* GA\_driverFuncs::GetPaletteData.  
     *See* GA\_VBEFuncs::GetPaletteData  
 GetPaletteDataExt. *See*  
     GA\_driverFuncs::GetPaletteDataExt  
 GetPixel. *See* GA\_2DRenderFuncs::GetPixel  
 GetPrimaryBuffer. *See*  
     GA\_bufferFuncs::GetPrimaryBuffer  
 GetUniqueFilename. *See*  
     GA\_initFuncs::GetUniqueFilename  
 GetVideoMode. *See* GA\_initFuncs::GetVideoMode  
 GetVideoModeInfo. *See*  
     GA\_initFuncs::GetVideoModeInfo  
 GetVideoModeInfoExt. *See*  
     GA\_initFuncs::GetVideoModeInfoExt  
 GetVSyncWidth. *See*  
     GA\_driverFuncs::GetVSyncWidth

## I

InitBuffers. *See* GA\_bufferFuncs::InitBuffers  
 InitDriver. *See* GA\_loaderFuncs::InitDriver  
 IsClipListChanged. *See*  
     GA\_clipperFuncs::IsClipListChanged  
 IsEmptyRegion. *See*  
     GA\_regionFuncs::IsEmptyRegion  
 IsEqualRegion. *See* GA\_regionFuncs::IsEqualRegion  
 IsHardwareCursor. *See*  
     GA\_cursorFuncs::IsHardwareCursor  
 IsIdle. *See* GA\_2DStateFuncs::IsIdle  
 IsVSync. *See* GA\_driverFuncs::IsVSync

## L

LockBuffer. *See* GA\_bufferFuncs::LockBuffer  
 LZTimerCount, 619, 624  
 LZTimerCountExt, 619, 620, 622, 624, 626  
 LZTimerLap, 621, 626  
 LZTimerLapExt, 620, 621, 622, 624, 626  
 LZTimerObject, 776  
 LZTimerOff, 620, 623, 626  
 LZTimerOffExt, 620, 622, 623, 624, 626  
 LZTimerOn, 620, 622, 625  
 LZTimerOnExt, 620, 622, 624, 625, 626

## M

MCS\_begin, 120, 123, 125  
 MCS\_beginExt, 120, 121  
 MCS\_controlsType, 122, 125, 126, 127, 130, 131,  
     133, 134, 563  
 MCS\_enableControl, 122, 125, 126, 127, 130, 131,  
     132, 133, 134  
 MCS\_end, 121, 123, 125  
 MCS\_getCapabilitiesString, 124  
 MCS\_getControlMax, 125  
 MCS\_getControlValue, 122, 125, 126, 127, 130, 131,  
     132, 133, 134  
 MCS\_getControlValues, 126, 127, 134  
 MCS\_getSelfTestReport, 128  
 MCS\_getTimingReport, 129, 565  
 MCS\_isControlSupported, 122, 125, 126, 127, 130,  
     131, 132, 133, 134  
 MCS\_polarityFlagsType, 129, 565  
 MCS\_resetControl, 122, 125, 126, 127, 130, 131, 132,  
     133, 134  
 MCS\_saveCurrentSettings, 122, 125, 126, 127, 130,  
     131, 132, 133, 134  
 MCS\_setControlValue, 122, 125, 126, 127, 130, 131,  
     132, 133, 134  
 MCS\_setControlValues, 126, 127, 133, 134  
 MDBX\_close, 135, 144  
 MDBX\_errCodes, 566  
 MDBX\_first, 136, 137, 141, 142, 143, 145, 146  
 MDBX\_flush, 137, 141, 146  
 MDBX\_getErrCode, 138, 139  
 MDBX\_getErrorMsg, 138, 139  
 MDBX\_importINF, 140  
 MDBX\_insert, 137, 141, 146

MDBX\_last, 136, 142, 143, 145  
 MDBG\_next, 136, 142, 143, 145  
 MDBG\_open, 135, 144  
 MDBG\_prev, 136, 142, 143, 145  
 MDBG\_update, 137, 141, 146

## N

N\_errorType, 492, 567  
 N\_fix32, 568  
 Nflt32, 569  
 N\_int16, 570  
 N\_int32, 571  
 N\_int8, 572  
 N\_physAddr, 573  
 N\_uint16, 574  
 N\_uint32, 575  
 N\_uint8, 576  
 NewRectRegion. *See*  
     GA\_regionFuncs::NewRectRegion  
 NewRegion. *See* GA\_regionFuncs::NewRegion

## O

OffsetRegion. *See* GA\_regionFuncs::OffsetRegion  
 OptimizeRegion. *See*  
     GA\_regionFuncs::OptimizeRegion

## P

PCI\_accessReg, 627, 628, 629, 630, 631, 780  
 PCI\_enumerate, 627, 628, 629, 630, 631  
 PCI\_getNumDevices, 627, 628, 629, 630, 631  
 PCI\_readRegBlock, 627, 628, 629, 630, 631  
 PCI\_writeRegBlock, 627, 628, 629, 630, 631  
 PCIAccessRegFlags, 627, 780  
 PCIAGPCapability, 777  
 PCIAGPCommand, 778  
 PCIAGPStatus, 779  
 PCICapsHeader, 781  
 PCICapsType, 781, 782  
 PCIClassTypes, 783  
 PCICommandFlags, 784, 785  
 PCIDeviceInfo, 627, 628, 784, 785, 787, 788, 789,  
     791, 793  
 PCIHeaderTypeFlags, 786, 787  
 PCIslot, 627, 794  
 PCIStatusFlags, 785, 788  
 PCIType0Info, 789  
 PCIType1Info, 791  
 PCIType2Info, 793  
 PE\_errorCodes, 577, 795  
 PE\_freeLibrary, 147, 150, 151, 152, 153, 632, 635,  
     636, 637, 638  
 PE\_getError, 148, 633  
 PE\_getFileSize, 149, 634  
 PE\_getProcAddress, 147, 148, 150, 151, 152, 153,  
     632, 633, 635, 636, 637, 638  
 PE\_loadLibrary, 147, 148, 150, 151, 152, 153, 632,  
     633, 635, 636, 637, 638  
 PE\_loadLibraryExt, 152, 637

PE\_loadLibraryMGL, 153, 638  
 PerformDisplaySwitch. *See*  
     GA\_initFuncs::PerformDisplaySwitch  
 PM\_agpCommitPhysical, 639, 641, 644  
 PM\_agpExit, 640, 642  
 PM\_agpFreePhysical, 639, 641  
 PM\_agpInit, 640, 642  
 PM\_agpMemoryType, 808  
 PM\_agpReleasePhysical, 643, 644  
 PM\_agpReservePhysical, 639, 642, 643, 644, 808  
 PM\_allocLockedMem, 645, 664  
 PM\_allocPage, 646, 665  
 PM\_allocRealSeg, 647, 650, 667, 700, 701, 712  
 PM\_backslash, 648  
 PM\_blockUntilTimeout, 649  
 PM\_calloc, 651, 662, 709, 718, 749  
 PM\_callRealMode, 650, 700, 701  
 PM\_closeConsole, 652, 673, 714, 721, 727, 740  
 PM\_doSuspendApp, 653  
 PM\_enableWriteCombine, 654, 711, 798, 799  
 PM\_enumWriteCombine, 655  
 PM\_enumWriteCombine\_t, 809  
 PM\_fatalCleanupHandler, 810  
 PM\_fatalError, 656, 730  
 PM\_findBPD, 657, 734  
 PM\_findClose, 658, 659, 660  
 PM\_findData, 800, 811  
 PM\_findFirstFile, 658, 659, 660  
 PM\_findNextFile, 658, 659, 660  
 PM\_flushTLB, 661  
 PM\_free, 651, 662, 709, 718, 749  
 PM\_freeLibrary, 663, 685, 705  
 PM\_freeLockedMem, 645, 664  
 PM\_freePage, 646, 665  
 PM\_freePhysicalAddr, 666, 711  
 PM\_freeRealSeg, 647, 667  
 PM\_freeShared, 668, 710  
 PM\_getA0000Pointer, 669  
 PM\_getBIOSPointer, 670  
 PM\_getBootDrive, 671  
 PM\_getch, 691  
 PM\_getCOMPort, 672  
 PM\_getConsoleStateSize, 652, 673, 714, 721, 727  
 PM\_getCurrentPath, 674, 692  
 PM\_getcwd, 674, 692  
 PM\_getDirectDrawWindow, 675, 704  
 PM\_getFileAttr, 676, 731  
 PM\_getFileTime, 677, 732  
 PM\_getIOPL, 678, 733  
 PM\_getLPTPort, 679  
 PM\_getMachineName, 680  
 PM\_getOSName, 681, 682  
 PM\_getOSType, 681, 682  
 PM\_getPhysicalAddr, 683, 684, 711  
 PM\_getPhysicalAddrRange, 683, 684  
 PM\_getProcAddress, 663, 685, 705  
 PM\_getSNAPConfigPath, 686, 687  
 PM\_getSNAPPPath, 686, 687  
 PM\_getUniqueID, 688  
 PM\_getVESABuf, 689  
 PM\_getVGAStateSize, 690, 724, 728

- PM\_haveBIOSAccess, 647, 650, 667, 670, 693, 700, 701, 712
  - PM\_HWND, 805
  - PM\_init, 694
  - PM\_inpb, 695, 696, 697, 715, 716, 717
  - PM\_inpd, 695, 696, 697
  - PM\_inpw, 695, 696, 697
  - PM\_installService, 698, 699
  - PM\_installServiceExt, 698, 699, 719, 743, 745
  - PM\_int86, 650, 693, 700, 701
  - PM\_int86x, 650, 700, 701
  - PM\_intHandler, 812
  - PM\_IRQHandle, 806
  - PM\_irqHandler, 813
  - PM\_isSDDActive, 702
  - PM\_kbhit, 703
  - PM\_loadDirectDraw, 675, 704, 746
  - PM\_loadLibrary, 663, 685, 705
  - PM\_lockCodePages, 618, 706, 707, 747
  - PM\_lockDataPages, 706, 707, 748
  - PM\_lockHandle, 814
  - PM\_makepath, 708
  - PM\_malloc, 651, 662, 709, 718, 749
  - PM\_mallocShared, 668, 710
  - PM\_mapPhysicalAddr, 666, 683, 684, 711
  - PM\_mapRealPointer, 647, 650, 667, 700, 701, 712
  - PM\_mkdir, 713, 725
  - PM\_MODULE, 807
  - PM\_openConsole, 652, 673, 714, 721, 727, 740
  - PM\_outpb, 695, 696, 697, 715, 716, 717
  - PM\_outpd, 715, 716, 717
  - PM\_outpw, 715, 716, 717
  - PM\_physAddr, 815
  - PM\_realloc, 651, 662, 709, 718, 749
  - PM\_removeService, 698, 699, 719, 743, 745
  - PM\_restartRealTimeClock, 720, 738, 739, 744
  - PM\_restoreConsoleState, 652, 673, 714, 721, 727
  - PM\_restoreRealTimeClockHandler, 720, 722, 738, 739, 744
  - PM\_restoreThreadPriority, 723, 735
  - PM\_restoreVGASState, 690, 724, 728
  - PM\_rmdir, 713, 725
  - PM\_runningInAWindow, 726
  - PM\_saveConsoleState, 652, 673, 714, 721, 727
  - PM\_saveVGASState, 690, 724, 728
  - PM\_setDebugLog, 729
  - PM\_setFatalErrorCleanup, 656, 730
  - PM\_setFileAttr, 676, 731, 800, 811
  - PM\_setFileTime, 677, 732, 819
  - PM\_setIOPL, 678, 733
  - PM\_setLocalBPDPath, 657, 734
  - PM\_setMaxThreadPriority, 735
  - PM\_setOSCursorLocation, 736
  - PM\_setOSScreenWidth, 737
  - PM\_setRealTimeClockFrequency, 720, 738, 739, 744
  - PM\_setRealTimeClockHandler, 720, 722, 738, 739, 744
  - PM\_setSuspendAppCallback, 673, 714, 721, 727, 740
  - PM\_sleep, 649, 741
  - PM\_splitpath, 742, 803
  - PM\_startService, 698, 699, 719, 743, 745
  - PM\_stopRealTimeClock, 720, 738, 739, 744
  - PM\_stopService, 698, 699, 719, 743, 745
  - PM\_suspendApp\_cb, 818
  - PM\_suspendAppCodesType, 816
  - PM\_suspendAppFlagsType, 817
  - PM\_time, 819
  - PM\_unloadDirectDraw, 675, 704, 746
  - PM\_unlockCodePages, 706, 747
  - PM\_unlockDataPages, 707, 748
  - PM\_useLocalMalloc, 651, 662, 709, 718, 749
  - PMBYTEREGS, 796
  - PMDWORDREGS, 797
  - PMEnableWriteCombineErrors, 798
  - PMEnableWriteCombineFlags, 654, 799
  - PMFileFlagsType, 676, 731, 800
  - PMREGS, 801, 820
  - PMSplitPathFlags, 742, 803
  - PMSREGS, 802, 821
  - PMWORDREGS, 804
  - PollForDisplaySwitch. *See* GA\_initFuncs::PollForDisplaySwitch
  - PostSwitchPhysicalResolution. *See* REF2D\_driver::PostSwitchPhysicalResolution
  - PtInRegion. *See* GA\_regionFuncs::PtInRegion
  - PutMonoImageLSBBM. *See* GA\_2DRenderFuncs::PutMonoImageLSBBM
  - PutMonoImageLSBLin. *See* GA\_2DRenderFuncs::PutMonoImageLSBLin
  - PutMonoImageLSBSys. *See* GA\_2DRenderFuncs::PutMonoImageLSBSys
  - PutMonoImageMSBBM. *See* GA\_2DRenderFuncs::PutMonoImageMSBBM
  - PutMonoImageMSBLin. *See* GA\_2DRenderFuncs::PutMonoImageMSBLin
  - PutMonoImageMSBSys. *See* GA\_2DRenderFuncs::PutMonoImageMSBSys
  - PutPixel. *See* GA\_2DRenderFuncs::PutPixel
- ## Q
- QueryFunctions. *See* REF2D\_driver::QueryFunctions.
  - See* GA\_loaderFuncs::QueryFunctions
- ## R
- REF2D\_driver, 155, 578
    - DrawRectExtSW, 579
    - ForceSoftwareOnly, 580
    - PostSwitchPhysicalResolution, 470, 581
    - QueryFunctions, 582
    - RotateBitmap, 583
    - SetColorCompareMask, 584
    - SetDrawBuffer, 292, 585, 586
    - SetDrawSurface, 585, 586
  - REF2D\_loadDriver, 154, 156, 578
  - REF2D\_queryFunctions, 88, 155, 582
  - REF2D\_unloadDriver, 154, 156
  - RMREGS, 820
  - RMSREGS, 821
  - RotateBitmap. *See* REF2D\_driver::RotateBitmap

**S**

SaveCRTCTimings. *See*  
     GA\_initFuncs::SaveCRTCTimings  
 SaveRestoreState. *See*  
     GA\_initFuncs::SaveRestoreState  
 SCIBegin. *See* GA\_SCIFuncs::SCIBegin  
 SCIdetect. *See* GA\_SCIFuncs::SCIdetect  
 SCILend. *See* GA\_SCIFuncs::SCILend  
 SCILoadSCL. *See* GA\_SCIFuncs::SCILoadSCL  
 SCILoadSDA. *See* GA\_SCIFuncs::SCILoadSDA  
 SCILoadSCL. *See* GA\_SCIFuncs::SCILoadSCL  
 SCILoadSDA. *See* GA\_SCIFuncs::SCILoadSDA  
 SectRegion. *See* GA\_regionFuncs::SectRegion  
 SectRegionRect. *See*  
     GA\_regionFuncs::SectRegionRect  
 Set8BitDAC. *See* GA\_VBEFuncs::Set8BitDAC  
 Set8x8ColorPattern. *See*  
     GA\_2DStateFuncs::Set8x8ColorPattern  
 Set8x8MonoPattern. *See*  
     GA\_2DStateFuncs::Set8x8MonoPattern  
 SetActiveBuffer. *See*  
     GA\_bufferFuncs::SetActiveBuffer  
 SetActiveHead. *See* GA\_initFuncs::SetActiveHead  
 SetAlphaValue. *See*  
     GA\_2DStateFuncs::SetAlphaValue  
 SetBackColor. *See* GA\_2DStateFuncs::SetBackColor  
 SetBank. *See* GA\_driverFuncs::SetBank  
 SetBlendFunc. *See* GA\_2DStateFuncs::SetBlendFunc  
 SetBytesPerLine. *See*  
     GA\_VBEFuncs::SetBytesPerLine  
 SetClipper. *See* GA\_bufferFuncs::SetClipper  
 SetColorCompareMask. *See*  
     REF2D\_driver::SetColorCompareMask  
 SetColorCursor. *See*  
     GA\_cursorFuncs::SetColorCursor  
 SetColorCursor256. *See*  
     GA\_cursorFuncs::SetColorCursor256  
 SetColorCursorRGB. *See*  
     GA\_cursorFuncs::SetColorCursorRGB  
 SetColorCursorRGBA. *See*  
     GA\_cursorFuncs::SetColorCursorRGBA  
 SetCRTCTimings. *See*  
     GA\_initFuncs::SetCRTCTimings  
 SetCursorPos. *See* GA\_cursorFuncs::SetCursorPos  
 SetCustomVideoMode. *See*  
     GA\_initFuncs::SetCustomVideoMode  
 SetDisplayOutput. *See*  
     GA\_initFuncs::SetDisplayOutput  
 SetDisplayStart. *See* GA\_driverFuncs::SetDisplayStart  
 SetDisplayStartXY. *See*  
     GA\_driverFuncs::SetDisplayStartXY  
 SetDrawBuffer. *See* REF2D\_driver::SetDrawBuffer.  
     *See* GA\_2DStateFuncs::SetDrawBuffer  
 SetDrawSurface. *See* REF2D\_driver::SetDrawSurface  
 SetForeColor. *See* GA\_2DStateFuncs::SetForeColor  
 SetGammaCorrectData. *See*  
     GA\_driverFuncs::SetGammaCorrectData  
 SetGammaCorrectDataExt. *See*  
     GA\_driverFuncs::SetGammaCorrectDataExt  
 SetGlobalRefresh. *See*  
     GA\_initFuncs::SetGlobalRefresh

SetLineStipple. *See*  
     GA\_2DStateFuncs::SetLineStipple  
 SetLineStippleCount. *See*  
     GA\_2DStateFuncs::SetLineStippleCount  
 SetLineStyle. *See* GA\_2DStateFuncs::SetLineStyle  
 SetMix. *See* GA\_2DStateFuncs::SetMix  
 SetModeProfile. *See* GA\_initFuncs::SetModeProfile  
 SetMonitorInfo. *See* GA\_initFuncs::SetMonitorInfo  
 SetMonoCursor. *See*  
     GA\_cursorFuncs::SetMonoCursor  
 SetMonoCursorColor. *See*  
     GA\_cursorFuncs::SetMonoCursorColor  
 SetOptions. *See* GA\_initFuncs::SetOptions  
 SetPaletteData. *See* GA\_driverFuncs::SetPaletteData.  
     *See* GA\_VBEFuncs::SetPaletteData  
 SetPaletteDataExt. *See*  
     GA\_driverFuncs::SetPaletteDataExt  
 SetPlaneMask. *See* GA\_2DStateFuncs::SetPlaneMask  
 SetRef2dPointer. *See* GA\_initFuncs::SetRef2dPointer  
 SetSoftwareRenderFuncs. *See*  
     GA\_initFuncs::SetSoftwareRenderFuncs  
 SetStereoDisplayStart. *See*  
     GA\_driverFuncs::SetStereoDisplayStart  
 SetVideoColorKey. *See*  
     GA\_videoFuncs::SetVideoColorKey  
 SetVideoMode. *See* GA\_initFuncs::SetVideoMode  
 SetVideoOutput. *See*  
     GA\_videoFuncs::SetVideoOutput  
 SetVSyncWidth. *See*  
     GA\_driverFuncs::SetVSyncWidth  
 ShowCursor. *See* GA\_cursorFuncs::ShowCursor  
 SrcTransBlt. *See* GA\_2DRenderFuncs::SrcTransBlt  
 SrcTransBltBM. *See*  
     GA\_2DRenderFuncs::SrcTransBltBM  
 SrcTransBltBuf. *See*  
     GA\_bufferFuncs::SrcTransBltBuf  
 SrcTransBltLin. *See*  
     GA\_2DRenderFuncs::SrcTransBltLin  
 SrcTransBltSys. *See*  
     GA\_2DRenderFuncs::SrcTransBltSys  
 StartVideoFrame. *See*  
     GA\_videoFuncs::StartVideoFrame  
 StretchBlt. *See* GA\_2DRenderFuncs::StretchBlt  
 StretchBltBM. *See*  
     GA\_2DRenderFuncs::StretchBltBM  
 StretchBltBuf. *See* GA\_bufferFuncs::StretchBltBuf  
 StretchBltLin. *See*  
     GA\_2DRenderFuncs::StretchBltLin  
 StretchBltSys. *See*  
     GA\_2DRenderFuncs::StretchBltSys  
 SwitchPhysicalResolution. *See*  
     GA\_initFuncs::SwitchPhysicalResolution

**T**

TraverseRegion. *See*  
     GA\_regionFuncs::TraverseRegion

**U**

ULZElapsedTime, 750, 751, 755, 756  
 ULZReadTime, 750, 751, 755, 756

ULZTimerCount, 752, 753, 754, 755, 756  
 ULZTimerLap, 752, 753, 754, 755  
 ULZTimerOff, 752, 753, 754, 755  
 ULZTimerOn, 752, 753, 754, 755  
 ULZTimerResolution, 750, 751, 752, 755, 756  
 UnionRegion. *See* GA\_regionFuncs::UnionRegion  
 UnionRegionOfs. *See*  
     GA\_regionFuncs::UnionRegionOfs  
 UnionRegionRect. *See*  
     GA\_regionFuncs::UnionRegionRect  
 UnloadDriver. *See* GA\_loaderFuncs::UnloadDriver  
 UnlockBuffer. *See* GA\_bufferFuncs::UnlockBuffer  
 UpdateCache. *See* GA\_bufferFuncs::UpdateCache  
 UpdateFromCache. *See*  
     GA\_bufferFuncs::UpdateFromCache  
 UpdateScreen. *See*  
     GA\_2DRenderFuncs::UpdateScreen  
 Use8x8ColorPattern. *See*  
     GA\_2DStateFuncs::Use8x8ColorPattern

Use8x8MonoPattern. *See*  
     GA\_2DStateFuncs::Use8x8MonoPattern  
 Use8x8TransColorPattern. *See*  
     GA\_2DStateFuncs::Use8x8TransColorPattern  
 Use8x8TransMonoPattern. *See*  
     GA\_2DStateFuncs::Use8x8TransMonoPattern

## W

WaitTillFlipped. *See*  
     GA\_bufferFuncs::WaitTillFlipped  
 WaitTillIdle. *See* GA\_2DStateFuncs::WaitTillIdle  
 WaitVSync. *See* GA\_driverFuncs::WaitVSync

## Z

ZTimerInit, 757  
 ZTimerInitExt, 757, 758