

SciTech MGL

Professional Multi-Platform Graphics Library

Reference Guide

Version 5.0

SciTech Software, Inc.
180 East 4th St, Suite 300
Chico, CA 95928

Main: (530) 894-8400
FAX: (530) 894-9069
www.scitechsoft.com

Information in the document is subject to change without notice. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of SciTech Software, Inc.

© 1991-2003 SciTech Software, Inc. All rights reserved.

SciTech Software, Inc.
180 East 4th St, Suite 300
Chico, CA 95928 USA
(530) 894-8400
(530) 894-9069 FAX

SciTech SNAP Graphics and SciTech MGL are trademarks of SciTech Software, Inc.

OpenGL is a registered trademark of Silicon Graphics, Inc.

All other marks are trademarks or registered trademarks of their respective companies.

Contents

MGL Library Overview	2
Environment detection and initialization	2
Device context creation and management	2
Device context information and manipulation	3
Color and palette manipulation	4
Viewport and clip rectangle manipulation	6
Double buffering support	6
Device clearing	7
Direct frame buffer access functions	7
Pixel plotting	7
Line drawing and clipping	7
Polyline and pixel drawing	8
Polygon drawing	8
Rectangle drawing	8
Ellipse drawing	8
Text attribute manipulation	9
Text drawing	9
Wide character text drawing	10
BitBlt support	10
Bitmap drawing support	10
Monochrome bitmap manipulation	11
Lightweight offscreen buffer support	11
Region management	12
Region algebra	12
RGB to 8/15/16 bit halftone dithering routines	13
Font loading and unloading functions	13
Obsolete bitmap font loading functions	13
Mouse cursor resource loading and unloading	14
Icon resource loading and unloading	14
Windows BMP bitmap loading, unloading and saving	14
PCX bitmap loading, unloading and saving	14
JPEG bitmap loading, unloading and saving	15
PNG bitmap loading, unloading and saving	15
Random number generation routines	15
Rectangle and Point manipulation	16
Window manager functions	16
OpenGL binding functions	17
Event handling	18
Mouse handling	19
Game Framework	19
Sprite Manager	20
Platform Specific: Windows	20

MGL Library Reference 23

External Functions.....	Error! Bookmark not defined.
CPU_getProcessorName	Error! Bookmark not defined.
CPU_getProcessorSpeed	Error! Bookmark not defined.
CPU_getProcessorSpeedInHZ	Error! Bookmark not defined.
CPU_getProcessorType.....	Error! Bookmark not defined.
CPU_have3DNow.....	Error! Bookmark not defined.
CPU_haveMMX	Error! Bookmark not defined.
CPU_haveRDTSC.....	Error! Bookmark not defined.
CPU_haveSSE.....	Error! Bookmark not defined.
EVT_allowLEDS.....	Error! Bookmark not defined.
EVT_asciiCode	Error! Bookmark not defined.
EVT_flush	Error! Bookmark not defined.
EVT_getCodePage.....	Error! Bookmark not defined.
EVT_getHeartBeatCallback	Error! Bookmark not defined.
EVT_getMousePos.....	Error! Bookmark not defined.
EVT_getNext	Error! Bookmark not defined.
EVT_halt	Error! Bookmark not defined.
EVT_isKeyDown.....	Error! Bookmark not defined.
EVT_joyIsPresent	Error! Bookmark not defined.
EVT_joySetCenter.....	Error! Bookmark not defined.
EVT_joySetLowerRight	Error! Bookmark not defined.
EVT_joySetUpperLeft	Error! Bookmark not defined.
EVT_peekNext	Error! Bookmark not defined.
EVT_pollJoystick.....	Error! Bookmark not defined.
EVT_post.....	Error! Bookmark not defined.
EVT_repeatCount	Error! Bookmark not defined.
EVT_scanCode	Error! Bookmark not defined.
EVT_setCodePage	Error! Bookmark not defined.
EVT_setHeartBeatCallback	Error! Bookmark not defined.
EVT_setMousePos	Error! Bookmark not defined.
EVT_setUserEventFilter.....	Error! Bookmark not defined.
GM_chooseMode.....	Error! Bookmark not defined.
GM_cleanup.....	Error! Bookmark not defined.
GM_exit	Error! Bookmark not defined.
GM_findMode.....	Error! Bookmark not defined.
GM_getDoDraw	Error! Bookmark not defined.
GM_getExitMainLoop	Error! Bookmark not defined.
GM_getHaveWin95.....	Error! Bookmark not defined.
GM_getHaveWinNT.....	Error! Bookmark not defined.
GM_init.....	Error! Bookmark not defined.
GM_initPath	Error! Bookmark not defined.
GM_initSysPalNoStatic	Error! Bookmark not defined.
GM_initWindowPos	Error! Bookmark not defined.
GM_mainLoop	Error! Bookmark not defined.
GM_processEvents.....	Error! Bookmark not defined.
GM_processEventsWin.....	Error! Bookmark not defined.
GM_realizePalette.....	Error! Bookmark not defined.
GM_registerEventProc	Error! Bookmark not defined.
GM_registerMainWindow.....	Error! Bookmark not defined.
GM_setAppActivate.....	Error! Bookmark not defined.
GM_setDrawFunc	Error! Bookmark not defined.
GM_setDriverOptions	Error! Bookmark not defined.

GM_setEventFunc	Error! Bookmark not defined.
GM_setExitFunc.....	Error! Bookmark not defined.
GM_setGameLogicFunc.....	Error! Bookmark not defined.
GM_setKeyDownFunc.....	Error! Bookmark not defined.
GM_setKeyRepeatFunc.....	Error! Bookmark not defined.
GM_setKeyUpFunc	Error! Bookmark not defined.
GM_setLeftBuffer.....	Error! Bookmark not defined.
GM_setMode.....	Error! Bookmark not defined.
GM_setModeExt.....	Error! Bookmark not defined.
GM_setModeFilterFunc.....	Error! Bookmark not defined.
GM_setModeSwitchFunc.....	Error! Bookmark not defined.
GM_setMouseDownFunc.....	Error! Bookmark not defined.
GM_setMouseMoveFunc.....	Error! Bookmark not defined.
GM_setMouseUpFunc.....	Error! Bookmark not defined.
GM_setPalette.....	Error! Bookmark not defined.
GM_setPreModeSwitchFunc.....	Error! Bookmark not defined.
GM_setRightBuffer.....	Error! Bookmark not defined.
GM_setSuspendAppCallback.....	Error! Bookmark not defined.
GM_startOpenGL.....	Error! Bookmark not defined.
GM_startStereo.....	Error! Bookmark not defined.
GM_stopStereo.....	Error! Bookmark not defined.
GM_swapBuffers.....	Error! Bookmark not defined.
GM_swapDirtyBuffers.....	Error! Bookmark not defined.
LZTimerCount	Error! Bookmark not defined.
LZTimerCountExt.....	Error! Bookmark not defined.
LZTimerLap	Error! Bookmark not defined.
LZTimerLapExt.....	Error! Bookmark not defined.
LZTimerOff.....	Error! Bookmark not defined.
LZTimerOffExt	Error! Bookmark not defined.
LZTimerOn	Error! Bookmark not defined.
LZTimerOnExt	Error! Bookmark not defined.
MGL_FixDiv.....	Error! Bookmark not defined.
MGL_FixMul.....	Error! Bookmark not defined.
MGL_FixMulDiv.....	Error! Bookmark not defined.
MGL_addCustomMode.....	Error! Bookmark not defined.
MGL_availableBitmap.....	Error! Bookmark not defined.
MGL_availableCursor.....	Error! Bookmark not defined.
MGL_availableFont.....	Error! Bookmark not defined.
MGL_availableIcon	Error! Bookmark not defined.
MGL_availableJPEG	Error! Bookmark not defined.
MGL_availablePCX	Error! Bookmark not defined.
MGL_availablePNG.....	Error! Bookmark not defined.
MGL_availablePages.....	Error! Bookmark not defined.
MGL_backfacing	Error! Bookmark not defined.
MGL_beginDirectAccess.....	Error! Bookmark not defined.
MGL_beginDirectAccessDC.....	Error! Bookmark not defined.
MGL_beginPaint.....	Error! Bookmark not defined.
MGL_beginPixel	Error! Bookmark not defined.
MGL_bitBlt	Error! Bookmark not defined.
MGL_bitBltCoord	Error! Bookmark not defined.
MGL_bitBltFx.....	Error! Bookmark not defined.
MGL_bitBltFxCoord	Error! Bookmark not defined.
MGL_bitBltPatt	Error! Bookmark not defined.

MGL_bitBltPattCoord.....	Error! Bookmark not defined.
MGL_buildMonoMask.....	Error! Bookmark not defined.
MGL_charWidth.....	Error! Bookmark not defined.
MGL_charWidth_W.....	Error! Bookmark not defined.
MGL_checkIdentityPalette.....	Error! Bookmark not defined.
MGL_clearDevice.....	Error! Bookmark not defined.
MGL_clearRegion.....	Error! Bookmark not defined.
MGL_clearViewport.....	Error! Bookmark not defined.
MGL_closeFontLib.....	Error! Bookmark not defined.
MGL_computePixelAddr.....	Error! Bookmark not defined.
MGL_copyBitmapToBuffer.....	Error! Bookmark not defined.
MGL_copyIntoRegion.....	Error! Bookmark not defined.
MGL_copyPage.....	Error! Bookmark not defined.
MGL_copyPageCoord.....	Error! Bookmark not defined.
MGL_copyRegion.....	Error! Bookmark not defined.
MGL_copyToBuffer.....	Error! Bookmark not defined.
MGL_createBuffer.....	Error! Bookmark not defined.
MGL_createCustomDC.....	Error! Bookmark not defined.
MGL_createDisplayDC.....	Error! Bookmark not defined.
MGL_createMemoryDC.....	Error! Bookmark not defined.
MGL_createOffscreenDC.....	Error! Bookmark not defined.
MGL_createScrollingDC.....	Error! Bookmark not defined.
MGL_createStereoDisplayDC.....	Error! Bookmark not defined.
MGL_createWindowedDC.....	Error! Bookmark not defined.
MGL_defRect.....	Error! Bookmark not defined.
MGL_defRectPt.....	Error! Bookmark not defined.
MGL_defaultAttributes.....	Error! Bookmark not defined.
MGL_defaultColor.....	Error! Bookmark not defined.
MGL_destroyBuffer.....	Error! Bookmark not defined.
MGL_destroyDC.....	Error! Bookmark not defined.
MGL_diffRegion.....	Error! Bookmark not defined.
MGL_diffRegionRect.....	Error! Bookmark not defined.
MGL_disableDriver.....	Error! Bookmark not defined.
MGL_disjointRect.....	Error! Bookmark not defined.
MGL_divotSize.....	Error! Bookmark not defined.
MGL_divotSizeCoord.....	Error! Bookmark not defined.
MGL_doubleBuffer.....	Error! Bookmark not defined.
MGL_drawGlyph.....	Error! Bookmark not defined.
MGL_drawRegion.....	Error! Bookmark not defined.
MGL_drawStr.....	Error! Bookmark not defined.
MGL_drawStrXY.....	Error! Bookmark not defined.
MGL_drawStrXY_W.....	Error! Bookmark not defined.
MGL_drawStr_W.....	Error! Bookmark not defined.
MGL_dstTransBlt.....	Error! Bookmark not defined.
MGL_dstTransBltCoord.....	Error! Bookmark not defined.
MGL_ellipse.....	Error! Bookmark not defined.
MGL_ellipseArc.....	Error! Bookmark not defined.
MGL_ellipseArcCoord.....	Error! Bookmark not defined.
MGL_ellipseArcEngine.....	Error! Bookmark not defined.
MGL_ellipseCoord.....	Error! Bookmark not defined.
MGL_ellipseEngine.....	Error! Bookmark not defined.
MGL_emptyRect.....	Error! Bookmark not defined.
MGL_emptyRegion.....	Error! Bookmark not defined.

<i>MGL_enableAllDrivers</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_enableOpenGLDrivers</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_endDirectAccess</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_endDirectAccessDC</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_endPaint</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_endPixel</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_enumerateFonts</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_equalPoint</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_equalRect</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_equalRegion</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_errorMsg</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_exit</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_fadePalette</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_fatalError</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_fclose</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_fillEllipse</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_fillEllipseArc</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_fillEllipseArcCoord</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_fillEllipseCoord</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_fillPolygon</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_fillPolygonCnvx</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_fillPolygonCnvxFX</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_fillPolygonFX</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_fillRect</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_fillRectCoord</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_fillRectPt</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_findMode</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_fopen</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_fread</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_freeRegion</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_fseek</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_ftell</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_fwrite</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_getActivePage</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_getAlphaValue</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_getArcCoords</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_getAspectRatio</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_getAttributes</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_getBackColor</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_getBackMode</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_getBitmapFromDC</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_getBitmapSize</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_getBitmapSizeExt</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_getBitsPerPixel</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_getBlendFunc</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_getCP</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_getCharMetrics</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_getCharMetrics_W</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_getClipRect</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_getClipRectDC</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_getClipRegion</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_getClipRegionDC</i>	<i>Error! Bookmark not defined.</i>
<i>MGL_getColor</i>	<i>Error! Bookmark not defined.</i>

MGL_getCurrentScanLine	Error! Bookmark not defined.
MGL_getDefaultPalette	Error! Bookmark not defined.
MGL_getDisplayStart.....	Error! Bookmark not defined.
MGL_getDitherMode.....	Error! Bookmark not defined.
MGL_getDivot	Error! Bookmark not defined.
MGL_getDivotCoord	Error! Bookmark not defined.
MGL_getDotsPerInch	Error! Bookmark not defined.
MGL_getFont.....	Error! Bookmark not defined.
MGL_getFontAntiAliasPalette	Error! Bookmark not defined.
MGL_getFontBlendMode.....	Error! Bookmark not defined.
MGL_getFontMetrics	Error! Bookmark not defined.
MGL_getFullScreenWindow	Error! Bookmark not defined.
MGL_getGammaRamp	Error! Bookmark not defined.
MGL_getGlyphHeight	Error! Bookmark not defined.
MGL_getGlyphWidth	Error! Bookmark not defined.
MGL_getHalfTonePalette	Error! Bookmark not defined.
MGL_getHardwareFlags.....	Error! Bookmark not defined.
MGL_getJPEGSize.....	Error! Bookmark not defined.
MGL_getJPEGSizeExt.....	Error! Bookmark not defined.
MGL_getLineStipple	Error! Bookmark not defined.
MGL_getLineStippleCount.....	Error! Bookmark not defined.
MGL_getLineStyle	Error! Bookmark not defined.
MGL_getPCXSize.....	Error! Bookmark not defined.
MGL_getPCXSizeExt	Error! Bookmark not defined.
MGL_getPNGSize	Error! Bookmark not defined.
MGL_getPNGSizeExt.....	Error! Bookmark not defined.
MGL_getPalette	Error! Bookmark not defined.
MGL_getPaletteEntry.....	Error! Bookmark not defined.
MGL_getPaletteSize.....	Error! Bookmark not defined.
MGL_getPaletteSnowLevel	Error! Bookmark not defined.
MGL_getPenBitmapPattern	Error! Bookmark not defined.
MGL_getPenPixmapPattern.....	Error! Bookmark not defined.
MGL_getPenPixmapTransparent.....	Error! Bookmark not defined.
MGL_getPenSize.....	Error! Bookmark not defined.
MGL_getPenStyle	Error! Bookmark not defined.
MGL_getPixel	Error! Bookmark not defined.
MGL_getPixelCoord	Error! Bookmark not defined.
MGL_getPixelCoordFast.....	Error! Bookmark not defined.
MGL_getPixelFast	Error! Bookmark not defined.
MGL_getPixelFormat	Error! Bookmark not defined.
MGL_getPlaneMask	Error! Bookmark not defined.
MGL_getPolygonType	Error! Bookmark not defined.
MGL_getSpaceExtra	Error! Bookmark not defined.
MGL_getTextDirection.....	Error! Bookmark not defined.
MGL_getTextJustify	Error! Bookmark not defined.
MGL_getTextSettings.....	Error! Bookmark not defined.
MGL_getTextSize	Error! Bookmark not defined.
MGL_getViewport	Error! Bookmark not defined.
MGL_getViewportDC.....	Error! Bookmark not defined.
MGL_getViewportOrg.....	Error! Bookmark not defined.
MGL_getViewportOrgDC.....	Error! Bookmark not defined.
MGL_getVisualPage	Error! Bookmark not defined.
MGL_getWriteMode	Error! Bookmark not defined.

MGL_getX	Error! Bookmark not defined.
MGL_getY	Error! Bookmark not defined.
MGL_glChooseVisual	Error! Bookmark not defined.
MGL_glCreateContext	Error! Bookmark not defined.
MGL_glDeleteContext	Error! Bookmark not defined.
MGL_glDisableMGLFuncs	Error! Bookmark not defined.
MGL_glEnableMGLFuncs	Error! Bookmark not defined.
MGL_glEnumerateDrivers	Error! Bookmark not defined.
MGL_glGetProcAddress	Error! Bookmark not defined.
MGL_glGetVisual	Error! Bookmark not defined.
MGL_glHaveHWOpenGL	Error! Bookmark not defined.
MGL_glMakeCurrent	Error! Bookmark not defined.
MGL_glRealizePalette	Error! Bookmark not defined.
MGL_glResizeBuffers	Error! Bookmark not defined.
MGL_glSetDriver	Error! Bookmark not defined.
MGL_glSetOpenGLType	Error! Bookmark not defined.
MGL_glSetPalette	Error! Bookmark not defined.
MGL_glSetVisual	Error! Bookmark not defined.
MGL_glSwapBuffers	Error! Bookmark not defined.
MGL_globalToLocal	Error! Bookmark not defined.
MGL_globalToLocalDC	Error! Bookmark not defined.
MGL_halfTonePixel	Error! Bookmark not defined.
MGL_halfTonePixel555	Error! Bookmark not defined.
MGL_halfTonePixel565	Error! Bookmark not defined.
MGL_haveWidePalette	Error! Bookmark not defined.
MGL_init	Error! Bookmark not defined.
MGL_insetRect	Error! Bookmark not defined.
MGL_isCurrentDC	Error! Bookmark not defined.
MGL_isDisplayDC	Error! Bookmark not defined.
MGL_isMemoryDC	Error! Bookmark not defined.
MGL_isOffscreenDC	Error! Bookmark not defined.
MGL_isOverlayDC	Error! Bookmark not defined.
MGL_isSimpleRegion	Error! Bookmark not defined.
MGL_isStereoDC	Error! Bookmark not defined.
MGL_isVSync	Error! Bookmark not defined.
MGL_isWindowedDC	Error! Bookmark not defined.
MGL_leftTop	Error! Bookmark not defined.
MGL_line	Error! Bookmark not defined.
MGL_lineCoord	Error! Bookmark not defined.
MGL_lineCoordExt	Error! Bookmark not defined.
MGL_lineEngine	Error! Bookmark not defined.
MGL_lineExt	Error! Bookmark not defined.
MGL_lineRel	Error! Bookmark not defined.
MGL_lineRelCoord	Error! Bookmark not defined.
MGL_lineTo	Error! Bookmark not defined.
MGL_lineToCoord	Error! Bookmark not defined.
MGL_loadBitmap	Error! Bookmark not defined.
MGL_loadBitmapExt	Error! Bookmark not defined.
MGL_loadBitmapIntoDC	Error! Bookmark not defined.
MGL_loadBitmapIntoDCExt	Error! Bookmark not defined.
MGL_loadCursor	Error! Bookmark not defined.
MGL_loadCursorExt	Error! Bookmark not defined.
MGL_loadFont	Error! Bookmark not defined.

MGL_loadFontExt	Error! Bookmark not defined.
MGL_loadFontInstance.....	Error! Bookmark not defined.
MGL_loadIcon.....	Error! Bookmark not defined.
MGL_loadIconExt.....	Error! Bookmark not defined.
MGL_loadJPEG.....	Error! Bookmark not defined.
MGL_loadJPEGExt.....	Error! Bookmark not defined.
MGL_loadJPEGIntoDC	Error! Bookmark not defined.
MGL_loadJPEGIntoDCExt	Error! Bookmark not defined.
MGL_loadPCX.....	Error! Bookmark not defined.
MGL_loadPCXExt.....	Error! Bookmark not defined.
MGL_loadPCXIntoDC	Error! Bookmark not defined.
MGL_loadPCXIntoDCExt.....	Error! Bookmark not defined.
MGL_loadPNG	Error! Bookmark not defined.
MGL_loadPNGExt	Error! Bookmark not defined.
MGL_loadPNGIntoDC.....	Error! Bookmark not defined.
MGL_loadPNGIntoDCExt	Error! Bookmark not defined.
MGL_localToGlobal	Error! Bookmark not defined.
MGL_localToGlobalDC	Error! Bookmark not defined.
MGL_lockBuffer.....	Error! Bookmark not defined.
MGL_lockToFrameRate	Error! Bookmark not defined.
MGL_makeCurrentDC	Error! Bookmark not defined.
MGL_mapToPalette	Error! Bookmark not defined.
MGL_maxCharWidth	Error! Bookmark not defined.
MGL_maxColor	Error! Bookmark not defined.
MGL_maxPage.....	Error! Bookmark not defined.
MGL_maxxx	Error! Bookmark not defined.
MGL_maxxDC.....	Error! Bookmark not defined.
MGL_maxy	Error! Bookmark not defined.
MGL_maxyDC.....	Error! Bookmark not defined.
MGL_memcpy.....	Error! Bookmark not defined.
MGL_memcpyVIRTDST.....	Error! Bookmark not defined.
MGL_memcpyVIRTSRC	Error! Bookmark not defined.
MGL_memset.....	Error! Bookmark not defined.
MGL_memsetl.....	Error! Bookmark not defined.
MGL_memsetw	Error! Bookmark not defined.
MGL_mirrorGlyph.....	Error! Bookmark not defined.
MGL_modeDriverName	Error! Bookmark not defined.
MGL_modeFlags	Error! Bookmark not defined.
MGL_modeResolution.....	Error! Bookmark not defined.
MGL_moveRel	Error! Bookmark not defined.
MGL_moveRelCoord.....	Error! Bookmark not defined.
MGL_moveTo.....	Error! Bookmark not defined.
MGL_moveToCoord.....	Error! Bookmark not defined.
MGL_newRegion	Error! Bookmark not defined.
MGL_offsetRect.....	Error! Bookmark not defined.
MGL_offsetRegion.....	Error! Bookmark not defined.
MGL_openFontLib	Error! Bookmark not defined.
MGL_openFontLibExt	Error! Bookmark not defined.
MGL_optimizeRegion	Error! Bookmark not defined.
MGL_packColor	Error! Bookmark not defined.
MGL_packColorExt	Error! Bookmark not defined.
MGL_packColorFast	Error! Bookmark not defined.
MGL_packColorFastExt.....	Error! Bookmark not defined.

MGL_pixel	Error! Bookmark not defined.
MGL_pixelCoord.....	Error! Bookmark not defined.
MGL_pixelCoordFast.....	Error! Bookmark not defined.
MGL_pixelFast.....	Error! Bookmark not defined.
MGL_polyLine	Error! Bookmark not defined.
MGL_polyPoint	Error! Bookmark not defined.
MGL_ptInRect	Error! Bookmark not defined.
MGL_ptInRectCoord	Error! Bookmark not defined.
MGL_ptInRegion	Error! Bookmark not defined.
MGL_ptInRegionCoord	Error! Bookmark not defined.
MGL_putBitmap.....	Error! Bookmark not defined.
MGL_putBitmapDstTrans.....	Error! Bookmark not defined.
MGL_putBitmapDstTransSection.....	Error! Bookmark not defined.
MGL_putBitmapFx.....	Error! Bookmark not defined.
MGL_putBitmapFxSection	Error! Bookmark not defined.
MGL_putBitmapMask	Error! Bookmark not defined.
MGL_putBitmapPatt	Error! Bookmark not defined.
MGL_putBitmapPattSection	Error! Bookmark not defined.
MGL_putBitmapSection	Error! Bookmark not defined.
MGL_putBitmapSrcTrans	Error! Bookmark not defined.
MGL_putBitmapSrcTransSection	Error! Bookmark not defined.
MGL_putBuffer.....	Error! Bookmark not defined.
MGL_putBufferDstTrans	Error! Bookmark not defined.
MGL_putBufferDstTransSection.....	Error! Bookmark not defined.
MGL_putBufferFx.....	Error! Bookmark not defined.
MGL_putBufferFxSection.....	Error! Bookmark not defined.
MGL_putBufferPatt	Error! Bookmark not defined.
MGL_putBufferPattSection	Error! Bookmark not defined.
MGL_putBufferSection	Error! Bookmark not defined.
MGL_putBufferSrcTrans	Error! Bookmark not defined.
MGL_putBufferSrcTransSection	Error! Bookmark not defined.
MGL_putDivot	Error! Bookmark not defined.
MGL_putIcon.....	Error! Bookmark not defined.
MGL_putMonoImage	Error! Bookmark not defined.
MGL_quickInit.....	Error! Bookmark not defined.
MGL_random.....	Error! Bookmark not defined.
MGL_randoml.....	Error! Bookmark not defined.
MGL_realColor	Error! Bookmark not defined.
MGL_realizePalette.....	Error! Bookmark not defined.
MGL_rect	Error! Bookmark not defined.
MGL_rectCoord	Error! Bookmark not defined.
MGL_rectPt	Error! Bookmark not defined.
MGL_registerEventProc	Error! Bookmark not defined.
MGL_registerFullScreenWindow	Error! Bookmark not defined.
MGL_restoreAttributes.....	Error! Bookmark not defined.
MGL_result.....	Error! Bookmark not defined.
MGL_resume.....	Error! Bookmark not defined.
MGL_rgbColor	Error! Bookmark not defined.
MGL_rgnEllipse.....	Error! Bookmark not defined.
MGL_rgnEllipseArc.....	Error! Bookmark not defined.
MGL_rgnGetArcCoords	Error! Bookmark not defined.
MGL_rgnLine	Error! Bookmark not defined.
MGL_rgnLineCoord.....	Error! Bookmark not defined.

MGL_rgnPolygon	Error! Bookmark not defined.
MGL_rgnPolygonCnvx	Error! Bookmark not defined.
MGL_rgnPolygonCnvxFX	Error! Bookmark not defined.
MGL_rgnPolygonFX	Error! Bookmark not defined.
MGL_rgnSolidEllipse	Error! Bookmark not defined.
MGL_rgnSolidEllipseArc	Error! Bookmark not defined.
MGL_rgnSolidRect	Error! Bookmark not defined.
MGL_rgnSolidRectCoord	Error! Bookmark not defined.
MGL_rgnSolidRectPt	Error! Bookmark not defined.
MGL_rightBottom.....	Error! Bookmark not defined.
MGL_rotateGlyph.....	Error! Bookmark not defined.
MGL_rotatePalette.....	Error! Bookmark not defined.
MGL_saveBitmapFromDC	Error! Bookmark not defined.
MGL_saveJPEGFromDC.....	Error! Bookmark not defined.
MGL_savePCXFromDC.....	Error! Bookmark not defined.
MGL_savePNGFromDC.....	Error! Bookmark not defined.
MGL_savePNGFromDCExt.....	Error! Bookmark not defined.
MGL_scanLine	Error! Bookmark not defined.
MGL_sectRect	Error! Bookmark not defined.
MGL_sectRectCoord	Error! Bookmark not defined.
MGL_sectRectFast	Error! Bookmark not defined.
MGL_sectRectFastCoord	Error! Bookmark not defined.
MGL_sectRegion.....	Error! Bookmark not defined.
MGL_sectRegionRect.....	Error! Bookmark not defined.
MGL_selectDisplayDevice	Error! Bookmark not defined.
MGL_setActivePage.....	Error! Bookmark not defined.
MGL_setAlphaValue.....	Error! Bookmark not defined.
MGL_setAspectRatio	Error! Bookmark not defined.
MGL_setBackColor	Error! Bookmark not defined.
MGL_setBackMode	Error! Bookmark not defined.
MGL_setBlendFunc	Error! Bookmark not defined.
MGL_setBlueCodeIndex	Error! Bookmark not defined.
MGL_setBufSize	Error! Bookmark not defined.
MGL_setClipRect.....	Error! Bookmark not defined.
MGL_setClipRectDC.....	Error! Bookmark not defined.
MGL_setClipRegion.....	Error! Bookmark not defined.
MGL_setClipRegionDC.....	Error! Bookmark not defined.
MGL_setColor	Error! Bookmark not defined.
MGL_setColorCI.....	Error! Bookmark not defined.
MGL_setColorRGB	Error! Bookmark not defined.
MGL_setDefaultPalette.....	Error! Bookmark not defined.
MGL_setDisplayStart	Error! Bookmark not defined.
MGL_setDitherMode	Error! Bookmark not defined.
MGL_setDotsPerInch	Error! Bookmark not defined.
MGL_setFileIO	Error! Bookmark not defined.
MGL_setFontAntiAliasPalette	Error! Bookmark not defined.
MGL_setFontBlendMode.....	Error! Bookmark not defined.
MGL_setGammaRamp.....	Error! Bookmark not defined.
MGL_setLineStipple	Error! Bookmark not defined.
MGL_setLineStippleCount	Error! Bookmark not defined.
MGL_setLineStyle.....	Error! Bookmark not defined.
MGL_setOpenGLFuncs	Error! Bookmark not defined.
MGL_setPalette.....	Error! Bookmark not defined.

MGL_setPaletteEntry	Error! Bookmark not defined.
MGL_setPaletteSnowLevel	Error! Bookmark not defined.
MGL_setPenBitmapPattern.....	Error! Bookmark not defined.
MGL_setPenPixmapPattern	Error! Bookmark not defined.
MGL_setPenPixmapTransparent	Error! Bookmark not defined.
MGL_setPenSize	Error! Bookmark not defined.
MGL_setPenStyle	Error! Bookmark not defined.
MGL_setPlaneMask	Error! Bookmark not defined.
MGL_setPolygonType.....	Error! Bookmark not defined.
MGL_setRelViewport	Error! Bookmark not defined.
MGL_setRelViewportDC.....	Error! Bookmark not defined.
MGL_setResult	Error! Bookmark not defined.
MGL_setSpaceExtra.....	Error! Bookmark not defined.
MGL_setSuspendAppCallback.....	Error! Bookmark not defined.
MGL_setTextDirection	Error! Bookmark not defined.
MGL_setTextEncoding	Error! Bookmark not defined.
MGL_setTextJustify.....	Error! Bookmark not defined.
MGL_setTextSettings	Error! Bookmark not defined.
MGL_setTextSize	Error! Bookmark not defined.
MGL_setViewport.....	Error! Bookmark not defined.
MGL_setViewportDC.....	Error! Bookmark not defined.
MGL_setViewportOrg	Error! Bookmark not defined.
MGL_setViewportOrgDC	Error! Bookmark not defined.
MGL_setVisualPage	Error! Bookmark not defined.
MGL_setWriteMode	Error! Bookmark not defined.
MGL_singleBuffer.....	Error! Bookmark not defined.
MGL_sizeX	Error! Bookmark not defined.
MGL_sizeY	Error! Bookmark not defined.
MGL_srand	Error! Bookmark not defined.
MGL_srcTransBlit	Error! Bookmark not defined.
MGL_srcTransBlitCoord	Error! Bookmark not defined.
MGL_startStereo.....	Error! Bookmark not defined.
MGL_stopStereo.....	Error! Bookmark not defined.
MGL_stretchBitmap.....	Error! Bookmark not defined.
MGL_stretchBitmapFx	Error! Bookmark not defined.
MGL_stretchBitmapFxSection.....	Error! Bookmark not defined.
MGL_stretchBitmapSection.....	Error! Bookmark not defined.
MGL_stretchBlit	Error! Bookmark not defined.
MGL_stretchBlitCoord.....	Error! Bookmark not defined.
MGL_stretchBlitFx	Error! Bookmark not defined.
MGL_stretchBlitFxCoord.....	Error! Bookmark not defined.
MGL_stretchBuffer	Error! Bookmark not defined.
MGL_stretchBufferFx	Error! Bookmark not defined.
MGL_stretchBufferFxSection	Error! Bookmark not defined.
MGL_stretchBufferSection.....	Error! Bookmark not defined.
MGL_surfaceAccessType	Error! Bookmark not defined.
MGL_suspend	Error! Bookmark not defined.
MGL_swapBuffers.....	Error! Bookmark not defined.
MGL_textBounds.....	Error! Bookmark not defined.
MGL_textBounds_W.....	Error! Bookmark not defined.
MGL_textHeight.....	Error! Bookmark not defined.
MGL_textWidth.....	Error! Bookmark not defined.
MGL_textWidth_W.....	Error! Bookmark not defined.

MGL_traverseRegion	Error! Bookmark not defined.
MGL_underScoreLocation	Error! Bookmark not defined.
MGL_underScoreLocation_W.....	Error! Bookmark not defined.
MGL_unionRect.....	Error! Bookmark not defined.
MGL_unionRectCoord.....	Error! Bookmark not defined.
MGL_unionRegion	Error! Bookmark not defined.
MGL_unionRegionOfs.....	Error! Bookmark not defined.
MGL_unionRegionRect	Error! Bookmark not defined.
MGL_unloadBitmap	Error! Bookmark not defined.
MGL_unloadCursor.....	Error! Bookmark not defined.
MGL_unloadFont.....	Error! Bookmark not defined.
MGL_unloadFontInstance	Error! Bookmark not defined.
MGL_unloadIcon	Error! Bookmark not defined.
MGL_unlockBuffer.....	Error! Bookmark not defined.
MGL_unpackColor.....	Error! Bookmark not defined.
MGL_unpackColorExt.....	Error! Bookmark not defined.
MGL_unpackColorFast	Error! Bookmark not defined.
MGL_unpackColorFastExt	Error! Bookmark not defined.
MGL_updateBufferCache.....	Error! Bookmark not defined.
MGL_updateFromBufferCache	Error! Bookmark not defined.
MGL_useFont	Error! Bookmark not defined.
MGL_usePenBitmapPattern.....	Error! Bookmark not defined.
MGL_usePenPixmapPattern	Error! Bookmark not defined.
MGL_vSync	Error! Bookmark not defined.
MGL_vecFontEngine.....	Error! Bookmark not defined.
MGL_wmBeginPaint.....	Error! Bookmark not defined.
MGL_wmCaptureEvents	Error! Bookmark not defined.
MGL_wmCoordGlobalToLocal	Error! Bookmark not defined.
MGL_wmCoordLocalToGlobal	Error! Bookmark not defined.
MGL_wmCreate.....	Error! Bookmark not defined.
MGL_wmCreateWindow	Error! Bookmark not defined.
MGL_wmDestroy	Error! Bookmark not defined.
MGL_wmDestroyWindow.....	Error! Bookmark not defined.
MGL_wmEndPaint.....	Error! Bookmark not defined.
MGL_wmGetRootWindow	Error! Bookmark not defined.
MGL_wmGetWindowAtPosition	Error! Bookmark not defined.
MGL_wmGetWindowFlags	Error! Bookmark not defined.
MGL_wmGetWindowParent.....	Error! Bookmark not defined.
MGL_wmGetWindowUserData	Error! Bookmark not defined.
MGL_wmInvalidateRect.....	Error! Bookmark not defined.
MGL_wmInvalidateRegion.....	Error! Bookmark not defined.
MGL_wmInvalidateWindow	Error! Bookmark not defined.
MGL_wmInvalidateWindowRect	Error! Bookmark not defined.
MGL_wmInvalidateWindowRegion	Error! Bookmark not defined.
MGL_wmLowerWindow.....	Error! Bookmark not defined.
MGL_wmPopGlobalEventHandler.....	Error! Bookmark not defined.
MGL_wmPopWindowEventHandler.....	Error! Bookmark not defined.
MGL_wmProcessEvent.....	Error! Bookmark not defined.
MGL_wmPushGlobalEventHandler.....	Error! Bookmark not defined.
MGL_wmPushWindowEventHandler.....	Error! Bookmark not defined.
MGL_wmRaiseWindow	Error! Bookmark not defined.
MGL_wmRemoveGlobalEventHandler	Error! Bookmark not defined.
MGL_wmRemoveWindowEventHandler	Error! Bookmark not defined.

MGL_wmReparentWindow	Error! Bookmark not defined.
MGL_wmSetGlobalCursor	Error! Bookmark not defined.
MGL_wmSetWindowCursor	Error! Bookmark not defined.
MGL_wmSetWindowDestructor	Error! Bookmark not defined.
MGL_wmSetWindowFlags	Error! Bookmark not defined.
MGL_wmSetWindowPainter	Error! Bookmark not defined.
MGL_wmSetWindowPosition	Error! Bookmark not defined.
MGL_wmSetWindowUserData	Error! Bookmark not defined.
MGL_wmShowWindow	Error! Bookmark not defined.
MGL_wmUncaptureEvents	Error! Bookmark not defined.
MGL_wmUpdateDC	Error! Bookmark not defined.
MS_getPos	Error! Bookmark not defined.
MS_hide	Error! Bookmark not defined.
MS_moveTo	Error! Bookmark not defined.
MS_obscure	Error! Bookmark not defined.
MS_setCursor	Error! Bookmark not defined.
MS_setCursorColor	Error! Bookmark not defined.
MS_setCursorColorExt	Error! Bookmark not defined.
MS_show	Error! Bookmark not defined.
SPR_destroyBitmap	Error! Bookmark not defined.
SPR_draw	Error! Bookmark not defined.
SPR_drawExt	Error! Bookmark not defined.
SPR_drawSection	Error! Bookmark not defined.
SPR_drawSectionExt	Error! Bookmark not defined.
SPR_mgrAddOpaqueBitmap	Error! Bookmark not defined.
SPR_mgrAddTransparentBitmap	Error! Bookmark not defined.
SPR_mgrEmpty	Error! Bookmark not defined.
SPR_mgrExit	Error! Bookmark not defined.
SPR_mgrInit	Error! Bookmark not defined.
ULZElapsedTime	Error! Bookmark not defined.
ULZReadTime	Error! Bookmark not defined.
ULZTimerCount	Error! Bookmark not defined.
ULZTimerLap	Error! Bookmark not defined.
ULZTimerOff	Error! Bookmark not defined.
ULZTimerOn	Error! Bookmark not defined.
ULZTimerResolution	Error! Bookmark not defined.
ZTimerInit	Error! Bookmark not defined.
ZTimerInitExt	Error! Bookmark not defined.
demo	Error! Bookmark not defined.
writeCursor	Error! Bookmark not defined.
Type Definitions	Error! Bookmark not defined.
CPU_largeInteger	Error! Bookmark not defined.
CPU_processorType	Error! Bookmark not defined.
EVT_asciiCodesType	Error! Bookmark not defined.
EVT_eventJoyAxisType	Error! Bookmark not defined.
EVT_eventJoyMaskType	Error! Bookmark not defined.
EVT_eventMaskType	Error! Bookmark not defined.
EVT_eventModMaskType	Error! Bookmark not defined.
EVT_eventMouseMaskType	Error! Bookmark not defined.
EVT_eventType	Error! Bookmark not defined.
EVT_masksType	Error! Bookmark not defined.
EVT_scanCodesType	Error! Bookmark not defined.
GMDC	Error! Bookmark not defined.

GM_driverOptions.....	Error! Bookmark not defined.
GM_modeFlagsType	Error! Bookmark not defined.
GM_modelInfo.....	Error! Bookmark not defined.
GM_stretchType.....	Error! Bookmark not defined.
LZTimerObject.....	Error! Bookmark not defined.
MGLBUF	Error! Bookmark not defined.
MGLDC	Error! Bookmark not defined.
MGL_Visual.....	Error! Bookmark not defined.
MGL_COLORS	Error! Bookmark not defined.
MGL_WIN_COLORS	Error! Bookmark not defined.
MGL_backModes.....	Error! Bookmark not defined.
MGL_bitBltFxFlagsType	Error! Bookmark not defined.
MGL_blendFuncType	Error! Bookmark not defined.
MGL_bufferFlagsType	Error! Bookmark not defined.
MGL_ditherModes	Error! Bookmark not defined.
MGL_errorType	Error! Bookmark not defined.
MGL_fontBlendType.....	Error! Bookmark not defined.
MGL_fontLibType.....	Error! Bookmark not defined.
MGL_fontType.....	Error! Bookmark not defined.
MGL_glContextFlagsType.....	Error! Bookmark not defined.
MGL_glOpenGLType	Error! Bookmark not defined.
MGL_hardwareFlagsType.....	Error! Bookmark not defined.
MGL_lineStyleType	Error! Bookmark not defined.
MGL_modeFlagsType	Error! Bookmark not defined.
MGL_palRotateType	Error! Bookmark not defined.
MGL_penStyleType	Error! Bookmark not defined.
MGL_polygonType	Error! Bookmark not defined.
MGL_refreshRateType	Error! Bookmark not defined.
MGL_rop3CodesType	Error! Bookmark not defined.
MGL_stereoBufType	Error! Bookmark not defined.
MGL_surfaceAccessFlagsType.....	Error! Bookmark not defined.
MGL_suspendAppCodesType.....	Error! Bookmark not defined.
MGL_suspendAppFlagsType.....	Error! Bookmark not defined.
MGL_textDirType	Error! Bookmark not defined.
MGL_textEncodingType.....	Error! Bookmark not defined.
MGL_textJustType.....	Error! Bookmark not defined.
MGL_waitVRTFlagType	Error! Bookmark not defined.
MGL_wmWindowFlags.....	Error! Bookmark not defined.
MGL_writeModeType.....	Error! Bookmark not defined.
M_int16.....	Error! Bookmark not defined.
M_int32.....	Error! Bookmark not defined.
M_int8.....	Error! Bookmark not defined.
M_uint16	Error! Bookmark not defined.
M_uint32	Error! Bookmark not defined.
M_uint8	Error! Bookmark not defined.
arc_coords_t.....	Error! Bookmark not defined.
attributes_t	Error! Bookmark not defined.
bitmap_t.....	Error! Bookmark not defined.
bltfx_t	Error! Bookmark not defined.
captureentry_t	Error! Bookmark not defined.
codepage_entry_t.....	Error! Bookmark not defined.
codepage_t	Error! Bookmark not defined.
color16_cursor_t.....	Error! Bookmark not defined.

<i>color256_cursor_t</i>	<i>Error! Bookmark not defined.</i>
<i>colorRGBA_cursor_t</i>	<i>Error! Bookmark not defined.</i>
<i>colorRGB_cursor_t</i>	<i>Error! Bookmark not defined.</i>
<i>color_t</i>	<i>Error! Bookmark not defined.</i>
<i>cursor_t</i>	<i>Error! Bookmark not defined.</i>
<i>event_t</i>	<i>Error! Bookmark not defined.</i>
<i>fileio_t</i>	<i>Error! Bookmark not defined.</i>
<i>fix32_t</i>	<i>Error! Bookmark not defined.</i>
<i>font_info_t</i>	<i>Error! Bookmark not defined.</i>
<i>font_lib_t</i>	<i>Error! Bookmark not defined.</i>
<i>font_t</i>	<i>Error! Bookmark not defined.</i>
<i>fxpoint_t</i>	<i>Error! Bookmark not defined.</i>
<i>globalevententry_t</i>	<i>Error! Bookmark not defined.</i>
<i>gmode_t</i>	<i>Error! Bookmark not defined.</i>
<i>icon_t</i>	<i>Error! Bookmark not defined.</i>
<i>metrics_t</i>	<i>Error! Bookmark not defined.</i>
<i>mono_cursor_t</i>	<i>Error! Bookmark not defined.</i>
<i>palette_ext_t</i>	<i>Error! Bookmark not defined.</i>
<i>palette_t</i>	<i>Error! Bookmark not defined.</i>
<i>pattern_t</i>	<i>Error! Bookmark not defined.</i>
<i>pixel_format_t</i>	<i>Error! Bookmark not defined.</i>
<i>pixpattern16_t</i>	<i>Error! Bookmark not defined.</i>
<i>pixpattern24_t</i>	<i>Error! Bookmark not defined.</i>
<i>pixpattern32_t</i>	<i>Error! Bookmark not defined.</i>
<i>pixpattern8_t</i>	<i>Error! Bookmark not defined.</i>
<i>pixpattern_t</i>	<i>Error! Bookmark not defined.</i>
<i>point_t</i>	<i>Error! Bookmark not defined.</i>
<i>rect_t</i>	<i>Error! Bookmark not defined.</i>
<i>region_t</i>	<i>Error! Bookmark not defined.</i>
<i>segment_t</i>	<i>Error! Bookmark not defined.</i>
<i>span_t</i>	<i>Error! Bookmark not defined.</i>
<i>text_settings_t</i>	<i>Error! Bookmark not defined.</i>
<i>window_t</i>	<i>Error! Bookmark not defined.</i>
<i>windowevententry_t</i>	<i>Error! Bookmark not defined.</i>
<i>winmng_t</i>	<i>Error! Bookmark not defined.</i>
Index	827

MGL Library Overview

This document contains the reference manual for the SciTech MGL Graphics Library and associated supplemental libraries. Please consult the [MGL Getting Started and Programmer's Guide](#) for more information on programming with the SciTech MGL.

All functions in the SciTech MGL work with an MGL device context (MGLDC), which can either be a specific device context passed to the function, or the currently active device context. All routines that take a specific MGLDC pointer, don't work with the values in the currently active device context, which means that these routines can be used to update a different device context than the currently active one. However if the device context passed is actually the currently active context, all the changes made to the specific device context will also be made to the currently active context as well.

Most of the routines in SciTech MGL are bound to the currently active device context, which can be changed with the MGL_makeCurrentDC function. When a context is made the currently active context, the values in the device context are *cached* in a special global data area for speed. Hence you should *not* change the values in a device context directly if that context is currently active, but should use the proper MGL functions to do so. When the active device context is changed to a new device context, the values in the currently cached device context are then flushed back to the original device context.

Because there is a lot of copying going on when changing active device context, this is something that you would not normally do between drawing individual primitives. Normally you will have one main device context for all your drawing output (either a memory device context or a display device context) that you will be drawing into, and this context will generally not change. The SciTech MGL will however not perform any copying if you attempt to set the active device context to the same device context. You can also set the active device context to nothing, by passing a NULL to MGL_makeCurrentDC.

Environment detection and initialization

The following functions are used to detect the installed hardware, initialize the SciTech MGL and obtain information about a particular device. Most of the functions listed below will be used by the application to tailor its use of the SciTech MGL functions depending on the underlying system configuration, and to provide system information to the end user.

MGL_addCustomMode	MGL_availablePages
MGL_disableDriver	MGL_enableAllDrivers
MGL_enableOpenGLDrivers	MGL_errorMsg
MGL_exit	MGL_fatalError
MGL_findMode	MGL_init
MGL_modeDriverName	MGL_modeFlags
MGL_modeResolution	MGL_quickInit
MGL_result	MGL_selectDisplayDevice
MGL_setBufSize	MGL_setResult

Device context creation and management

The following functions are used to create and destroy device context. You must create a an MGL device context before you can do any drawing.

MGL_createCustomDC	MGL_createDisplayDC
MGL_createMemoryDC	MGL_createOffscreenDC
MGL_createScrollingDC	MGL_createStereoDisplayDC
MGL_createWindowedDC	MGL_destroyDC
MGL_getHardwareFlags	MGL_isDisplayDC
MGL_isMemoryDC	MGL_isOffscreenDC
MGL_isOverlayDC	MGL_isStereoDC
MGL_isWindowedDC	MGL_surfaceAccessType

Device context information and manipulation

The following functions provide the API routines that are used to change the attributes of the currently active device context, such as the current foreground and background colors, and the current pen pattern. These routines also provide information about device contexts, such as the resolution, pixel depth, aspect ratio etc.

Note that some of these functions are bound to a specific device context, and if this device context is not the currently active context, the active context will not be affected.

MGL_computePixelAddr	MGL_defaultAttributes
MGL_getAlphaValue	MGL_getAspectRatio
MGL_getAttributes	MGL_getBackMode
MGL_getBitsPerPixel	MGL_getBlendFunc
MGL_getFontBlendMode	MGL_getLineStipple
MGL_getLineStippleCount	MGL_getLineStyle
MGL_getPenBitmapPattern	MGL_getPenPixmapPattern
MGL_getPenPixmapTransparent	MGL_getPenSize
MGL_getPenStyle	MGL_getPixelFormat
MGL_getPlaneMask	MGL_getPolygonType
MGL_isCurrentDC	MGL_makeCurrentDC
MGL_maxPage	MGL_restoreAttributes
MGL_setAlphaValue	MGL_setAspectRatio
MGL_setBackMode	MGL_setBlendFunc
MGL_setFontBlendMode	MGL_setLineStipple
MGL_setLineStippleCount	MGL_setLineStyle
MGL_setPenBitmapPattern	MGL_setPenPixmapPattern
MGL_setPenPixmapTransparent	MGL_setPenSize
MGL_setPenStyle	MGL_setPlaneMask
MGL_setPolygonType	MGL_setWriteMode
MGL_sizeX	MGL_sizeY
MGL_usePenBitmapPattern	MGL_usePenPixmapPattern

Color and palette manipulation

The following functions provide full control over colors in the SciTech MGL. SciTech MGL can be running in either color index modes (4 and 8 bits per pixel) or RGB modes (15 bits and above modes, and also RGB 8 bit dithered modes). In color index modes the final color of a pixel is determined by the color lookup table or palette associated with the device context. SciTech MGL provides functions for setting and retrieving color values in the color lookup tables for a device context, and for realizing the color palette on hardware devices.

Note that *all* device contexts have a color lookup table even for memory device contexts and RGB display device contexts. SciTech MGL will convert color values in 4 and 8 bit memory devices on the fly when BitBlt'ing to any other display device context with 4 bits or more of pixel depth. When doing a BitBlt operation from a color index device context to another color index device context, SciTech MGL will find the closest color if an exact match is not found (this can be turned off for speed though). When doing a BitBlt operation from a color index device context to an RGB device context, the color lookup table in the destination RGB device context will be used to convert the color values.

In order for maximum BitBlt performance when the destination device context is in a color index mode, you must ensure that the color table in the source device context is identical to the color table in the destination device context, in which case no bitmap translation will be performed (i.e.: SciTech MGL determines there is an identity palette mapping). If you know that all BitBlt operations will be performed have identity palettes, you can use the function `MGL_checkIdentityPalette` to turn on and off this checking code to obtain higher performance. Note however that this does *not* turn off identity palette checking when BitBlt'ing to a windowed device context under Windows, as this cannot be turned off.

Note also that all SciTech MGL color values are passed to the SciTech MGL as either a color index or a packed RGB value, packed for the correct format expected by the device context. Rather than setting the color value directory, you can use the utility functions `MGL_setColorCI` and `MGL_setColorRGB` to set the color given a color index or an RGB color value. When running in RGB modes, the `MGL_setColorCI` will convert the color index to the proper SciTech MGL color value using the device's color lookup table. When running in color index modes, the `MGL_setColorRGB` will set the color to the color index with the color palette entry that is the closest to the passed in RGB color value. Functions and macros are provided to pack RGB color values into the proper SciTech MGL color values to be passed to the SciTech MGL. The `MGL_pack*` family of functions is used to pack color values from 8 bit RGB tuples into the proper SciTech MGL packed RGB color values.

<code>MGL_checkIdentityPalette</code>	<code>MGL_defaultColor</code>
<code>MGL_fadePalette</code>	<code>MGL_getBackColor</code>
<code>MGL_getColor</code>	<code>MGL_getDefaultPalette</code>
<code>MGL_getDitherMode</code>	<code>MGL_getFontAntiAliasPalette</code>
<code>MGL_getGammaRamp</code>	<code>MGL_getPalette</code>
<code>MGL_getPaletteEntry</code>	<code>MGL_getPaletteSize</code>
<code>MGL_getPaletteSnowLevel</code>	<code>MGL_getWriteMode</code>
<code>MGL_haveWidePalette</code>	<code>MGL_mapToPalette</code>
<code>MGL_maxColor</code>	<code>MGL_packColor</code>
<code>MGL_packColorExt</code>	<code>MGL_packColorFast</code>
<code>MGL_packColorFastExt</code>	<code>MGL_realColor</code>
<code>MGL_realizePalette</code>	<code>MGL_rgbColor</code>
<code>MGL_rotatePalette</code>	<code>MGL_setBackColor</code>
<code>MGL_setColor</code>	<code>MGL_setColorCI</code>
<code>MGL_setColorRGB</code>	<code>MGL_setDefaultPalette</code>
<code>MGL_setDitherMode</code>	<code>MGL_setFontAntiAliasPalette</code>
<code>MGL_setGammaRamp</code>	<code>MGL_setPalette</code>
<code>MGL_setPaletteEntry</code>	<code>MGL_setPaletteSnowLevel</code>
<code>MGL_unpackColor</code>	<code>MGL_unpackColorExt</code>
<code>MGL_unpackColorFast</code>	<code>MGL_unpackColorFastExt</code>

Viewport and clip rectangle manipulation

The following functions provide the ability to change the current viewport that is used to display all subsequent drawing operations, and to change the current clipping rectangle or clipping region within the current viewport. The clip rectangle or clip region is always set in local viewport coordinates, and can never be larger than the current viewport (it will be clipped to the viewport boundary). Complex clip regions are fully supported and allow the SciTech MGL to clip output to a non-rectangle area of the screen. This is primarily used for GUI and window manager libraries.

Also provided are functions for converting coordinates between global screen coordinates and local viewport coordinates, and for obtaining the dimensions of the current viewport.

MGL_getClipRect	MGL_getClipRectDC
MGL_getClipRegion	MGL_getClipRegionDC
MGL_getViewport	MGL_getViewportDC
MGL_getViewportOrg	MGL_getViewportOrgDC
MGL_globalToLocal	MGL_globalToLocalDC
MGL_localToGlobal	MGL_localToGlobalDC
MGL_maxx	MGL_maxxDC
MGL_maxy	MGL_maxyDC
MGL_setClipRect	MGL_setClipRectDC
MGL_setClipRegion	MGL_setClipRegionDC
MGL_setRelViewport	MGL_setRelViewportDC
MGL_setViewport	MGL_setViewportDC
MGL_setViewportOrg	MGL_setViewportOrgDC

Double buffering support

The following functions provide support for hardware double buffering on display devices that have two display buffers. Hardware double buffering is used to achieve flicker free animation, but drawing to a hidden portion of display memory and then instantly updating the screen by changing the visible display page to the previously hidden display page. You can also use the MGL_setDisplayStart function to scroll around within a hardware scrolling or panning device context. For combining hardware scrolling and double buffering, you can use MGL_setDisplayStart and MGL_setVisualPage together.

MGL_doubleBuffer	MGL_getActivePage
MGL_getCurrentScanLine	MGL_getDisplayStart
MGL_getVisualPage	MGL_isVSync
MGL_lockToFrameRate	MGL_setActivePage
MGL_setDisplayStart	MGL_setVisualPage
MGL_singleBuffer	MGL_swapBuffers
MGL_vSync	

Device clearing

The following functions provide for clearing the entire device or just the current viewport to the background color.

MGL_clearDevice	MGL_clearViewport
-----------------	-------------------

Direct frame buffer access functions

The following functions are used to allow the application to get direct access to the drawing surface of a device context for custom rendering operations.

MGL_beginDirectAccess	MGL_beginDirectAccessDC
MGL_endDirectAccess	MGL_endDirectAccessDC

Pixel plotting

The following functions provide support for displaying single pixels, and for retrieving the color of single pixels for a device context. The *Fast* style functions require that you call the MGL_beginPixel function to set the hardware for drawing multiple pixels as fast as possible.

MGL_beginPixel	MGL_endPixel
MGL_getPixel	MGL_getPixelCoord
MGL_getPixelCoordFast	MGL_getPixelFast
MGL_pixel	MGL_pixelCoord
MGL_pixelCoordFast	MGL_pixelFast

Line drawing and clipping

The following functions provide support for drawing lines. SciTech MGL provides support for rasterizing lines with integer endpoints.

MGL_getCP	MGL_getX
MGL_getY	MGL_line
MGL_lineCoord	MGL_lineCoordExt
MGL_lineEngine	MGL_lineExt
MGL_lineRel	MGL_lineRelCoord
MGL_lineTo	MGL_lineToCoord
MGL_moveRel	MGL_moveRelCoord
MGL_moveTo	MGL_moveToCoord
MGL_scanLine	

Polyline and pixel drawing

The following functions provide support for drawing sets of integer coordinate lines and pixels.

MGL_polyLine	MGL_polyPoint
--------------	---------------

Polygon drawing

The following polygon routines come in two flavors. One takes vertex coordinates in integer format (16 or 32 bit integers depending on memory model) and the other takes vertex coordinates in 16.16 fixed-point format. The fixed-point format version is always the fastest, and in fact the integer versions simply convert all the vertices to fixed points and call the fixed-point routine.

MGL_fillPolygon	MGL_fillPolygonCnvx
MGL_fillPolygonCnvxFX	MGL_fillPolygonFX

Rectangle drawing

The following functions provide support for drawing outlined rectangles and solid rectangles filled with the current pen pattern.

MGL_fillRect	MGL_fillRectCoord
MGL_fillRectPt	MGL_rect
MGL_rectCoord	MGL_rectPt

Ellipse drawing

The following functions provide support for drawing ellipses and elliptical arcs, either as outlines or solid filled with the current pen pattern. Note that SciTech MGL provides

the ability to draw ellipses given a center coordinate and integer radii, or by providing a bounding rectangle for the ellipse. The bounding rectangle versions allow you to draw any sized ellipse, some of which will have the center coordinate located on a non-integer half-pixel boundary.

MGL_ellipse	MGL_ellipseArc
MGL_ellipseArcCoord	MGL_ellipseArcEngine
MGL_ellipseCoord	MGL_ellipseEngine
MGL_fillEllipse	MGL_fillEllipseArc
MGL_fillEllipseArcCoord	MGL_fillEllipseCoord
MGL_getArcCoords	

Text attribute manipulation

The following functions provide the ability to control text rasterizing attributes such as the horizontal and vertical justification, text direction and text size (for vector fonts only). Also provided are functions for measuring the size of text in the current font for appropriate character placement computations.

MGL_charWidth	MGL_getCharMetrics
MGL_getFontMetrics	MGL_getSpaceExtra
MGL_getTextDirection	MGL_getTextJustify
MGL_getTextSettings	MGL_getTextSize
MGL_maxCharWidth	MGL_setSpaceExtra
MGL_setTextDirection	MGL_setTextEncoding
MGL_setTextJustify	MGL_setTextSettings
MGL_setTextSize	MGL_textBounds
MGL_textHeight	MGL_textWidth
MGL_underScoreLocation	

Text drawing

The following functions provide the ability to draw text in the current text attributes, such as the justification and direction. The MGL_vecFontEngine function allows you to rasterize vector fonts by calling your own drawing routines, which can be used to do things like pass the vertices for vector fonts through 2D and 3D transformations to draw 2D and 3D transformed text.

MGL_drawStr	MGL_drawStrXY
MGL_getFont	MGL_useFont
MGL_vecFontEngine	

Wide character text drawing

The following functions provide the ability to draw wide character text in the current text attributes, such as the justification and direction. Wide character text fonts are useful for supporting output of Asian characters using Asian fonts. The SciTech MGL generally only supports wide character fonts with TrueType fonts, since the regular bitmap fonts only support 8-bit characters.

MGL_charWidth_W	MGL_drawStrXY_W
MGL_drawStr_W	MGL_getCharMetrics_W
MGL_textBounds_W	MGL_textWidth_W
MGL_underScoreLocation_W	

BitBlt support

The following functions provide support for copying blocks of image data between device contexts, or between areas of the same device context. The data can be copied using a simple solid copy, with arbitrary stretching and shrinking of the data to a different-sized destination rectangle and with source transparency for drawing transparent sprites for animation. The SciTech MGL also fully supports rasterizing of monochrome bitmaps, which can be used for fast masking operations and for rasterizing custom monochrome text.

MGL_bitBlt	MGL_bitBltCoord
MGL_bitBltFx	MGL_bitBltFxCoord
MGL_bitBltPatt	MGL_bitBltPattCoord
MGL_copyPage	MGL_copyPageCoord
MGL_divotSize	MGL_divotSizeCoord
MGL_dstTransBlt	MGL_dstTransBltCoord
MGL_getDivot	MGL_getDivotCoord
MGL_putDivot	MGL_putMonoImage
MGL_srcTransBlt	MGL_srcTransBltCoord
MGL_stretchBlt	MGL_stretchBltCoord
MGL_stretchBltFx	MGL_stretchBltFxCoord

Bitmap drawing support

The following functions provide support for copy lightweight bitmap image data to device contexts. The data can be copied using a simple solid copy, with arbitrary stretching and shrinking of the data to a different-sized destination rectangle and with source transparency for drawing transparent sprites for animation. You can also draw just a section of the bitmap on the device context as well. The SciTech MGL also fully

supports rasterizing of monochrome bitmaps, which can be used for fast masking operations and for rasterizing custom monochrome text.

MGL_putBitmap	MGL_putBitmapDstTrans
MGL_putBitmapDstTransSection	MGL_putBitmapFx
MGL_putBitmapFxSection	MGL_putBitmapMask
MGL_putBitmapPatt	MGL_putBitmapPattSection
MGL_putBitmapSection	MGL_putBitmapSrcTrans
MGL_putBitmapSrcTransSection	MGL_putIcon
MGL_stretchBitmap	MGL_stretchBitmapFx
MGL_stretchBitmapFxSection	MGL_stretchBitmapSection

Monochrome bitmap manipulation

The following functions provide support for drawing monochrome glyphs (used to implement custom text rasterizing routines) and performing rotations and other operations on monochrome bitmaps.

MGL_drawGlyph	MGL_getGlyphHeight
MGL_getGlyphWidth	MGL_mirrorGlyph
MGL_rotateGlyph	

Lightweight offscreen buffer support

The following functions provide support for creating, destroying and copying the contents of lightweight buffers to device contexts. Lightweight buffers may exist in system memory or hardware video memory. They are like a device context in that they can exist in video memory, but they have no rendering state associated with them so you cannot draw on lightweight buffers directly. Generally lightweight buffers are filled with bitmap data and used for fast sprite animation.

The data can be copied using a simple solid copy, with arbitrary stretching and shrinking of the data to a different-sized destination rectangle and with source transparency for drawing transparent sprites for animation. You can also draw just a section of the lightweight buffer on the device context as well.

MGL_copyBitmapToBuffer	MGL_copyToBuffer
MGL_createBuffer	MGL_destroyBuffer
MGL_lockBuffer	MGL_putBuffer
MGL_putBufferDstTrans	MGL_putBufferDstTransSection
MGL_putBufferFx	MGL_putBufferFxSection
MGL_putBufferPatt	MGL_putBufferPattSection
MGL_putBufferSection	MGL_putBufferSrcTrans
MGL_putBufferSrcTransSection	MGL_stretchBuffer
MGL_stretchBufferFx	MGL_stretchBufferFxSection
MGL_stretchBufferSection	MGL_unlockBuffer
MGL_updateBufferCache	MGL_updateFromBufferCache

Region management

The following functions provide support for managing complex regions, and generating new complex region primitives. Complex regions are used to represent 2D arbitrarily complex regions as unions of smaller rectangles, and can represent shapes with complex outlines, holes in the middle and even totally disjoint areas. These routines allow you to create, copy, free and draw such regions, as well as generate regions with specific shapes that can be combined with other regions to produce more complex shapes. Once you have created a complex region, you can make that region the clip region for a device context, allowing you to perform complex clipping of drawing operations on a device context. You can also traverse a complex region, which allows you to call a particular function for every rectangle in the union of rectangles that make up the complex region.

MGL_clearRegion	MGL_copyIntoRegion
MGL_copyRegion	MGL_drawRegion
MGL_freeRegion	MGL_isSimpleRegion
MGL_newRegion	MGL_rgnEllipse
MGL_rgnEllipseArc	MGL_rgnGetArcCoords
MGL_rgnLine	MGL_rgnLineCoord
MGL_rgnPolygon	MGL_rgnPolygonCnvx
MGL_rgnPolygonCnvxFX	MGL_rgnPolygonFX
MGL_rgnSolidEllipse	MGL_rgnSolidEllipseArc
MGL_rgnSolidRect	MGL_rgnSolidRectCoord
MGL_rgnSolidRectPt	MGL_traverseRegion

Region algebra

The following functions provide support for performing Boolean arithmetic operations on complex regions to produce new complex regions. These Boolean arithmetic routines form the heart of the SciTech MGL complex clip region support, and allow you to

construct arbitrarily complex shapes from the simple region primitives that the SciTech MGL can generate for you.

MGL_diffRegion	MGL_diffRegionRect
MGL_emptyRegion	MGL_equalRegion
MGL_offsetRegion	MGL_optimizeRegion
MGL_ptInRegionCoord	MGL_sectRegion
MGL_sectRegionRect	MGL_unionRegion
MGL_unionRegionOfs	MGL_unionRegionRect

RGB to 8/15/16 bit halftone dithering routines

The following functions allow you to obtain a copy of the SciTech MGL halftone palette, which is used for RGB to 8-bit real time dithering, and for performing the dither operation on a single pixel so you can implement your own real-time dithering routines. You can also do fast halftone dithering on 15-bit (5:5:5) and 16-bit (5:6:5) format pixels.

MGL_getHalfTonePalette	MGL_halfTonePixel
MGL_halfTonePixel555	MGL_halfTonePixel565

Font loading and unloading functions

The following functions provide support for loading and unloading SciTech MGL font resources from font libraries. The SciTech MGL can load standard Windows font libraries as well as TrueType font libraries.

MGL_closeFontLib	MGL_enumerateFonts
MGL_getDotsPerInch	MGL_loadFontInstance
MGL_openFontLib	MGL_openFontLibExt
MGL_setDotsPerInch	MGL_unloadFontInstance

Obsolete bitmap font loading functions

The following functions provide support for loading and unloading SciTech MGL font resources in the Windows font file format. The fonts are stored individually with one font per file. These functions are provided for compatibility with older versions of the SciTech MGL. You should use the newer font library code instead so you can store multiple font sizes in a single file.

MGL_availableFont	MGL_loadFont
MGL_loadFontExt	MGL_unloadFont

Mouse cursor resource loading and unloading

The following functions provide support for loading and unloading SciTech MGL mouse cursor resources. The SciTech MGL can load standard Windows style cursor files.

MGL_availableCursor	MGL_loadCursor
MGL_loadCursorExt	MGL_unloadCursor

Icon resource loading and unloading

The following functions provide support for loading and unloading SciTech MGL icon resources. The SciTech MGL can load standard Windows style icon files.

MGL_availableIcon	MGL_loadIcon
MGL_loadIconExt	MGL_unloadIcon

Windows BMP bitmap loading, unloading and saving

The following functions provide support for loading and unloading SciTech MGL bitmap resources in the Windows bitmap format (BMP). The SciTech MGL can load standard Windows style bitmap files in all formats supported by Windows. The SciTech MGL can also load and save Windows bitmap files in all of the color depths supported by the SciTech MGL, including 15-bit and 16-bit formats. Those bitmaps formats are not supported directly by Windows or Windows bitmap editing programs. The bitmap data will be converted to the format of the destination device context if the bitmap is loaded into a device context that is of a different pixel format to the incoming bitmap.

MGL_availableBitmap	MGL_buildMonoMask
MGL_getBitmapFromDC	MGL_getBitmapSize
MGL_getBitmapSizeExt	MGL_loadBitmap
MGL_loadBitmapExt	MGL_loadBitmapIntoDC
MGL_loadBitmapIntoDCExt	MGL_saveBitmapFromDC
MGL_unloadBitmap	

PCX bitmap loading, unloading and saving

The following functions provide support for loading and unloading SciTech MGL bitmap resources in the PCX format. The SciTech MGL can load and save PCX files in 1, 4 and 8-bits per formats. The bitmap data will be converted to the format of the destination device context if the bitmap is loaded into a device context that is of a different pixel format to the incoming bitmap.

MGL_availablePCX	MGL_getPCXSize
MGL_getPCXSizeExt	MGL_loadPCX
MGL_loadPCXExt	MGL_loadPCXIntoDC
MGL_loadPCXIntoDCExt	MGL_savePCXFromDC

JPEG bitmap loading, unloading and saving

The following functions provide support for loading and unloading SciTech MGL bitmap resources in the 24-bit JPEG format. The SciTech MGL can load and save 24-bit JPEG files from any device context. The bitmap data will be converted to the format of the destination device context if the bitmap is loaded into a device context that is of a different pixel format to the incoming bitmap.

MGL_availableJPEG	MGL_getJPEGSize
MGL_getJPEGSizeExt	MGL_loadJPEG
MGL_loadJPEGExt	MGL_loadJPEGIntoDC
MGL_loadJPEGIntoDCExt	MGL_saveJPEGFromDC

PNG bitmap loading, unloading and saving

The following functions provide support for loading and unloading SciTech MGL bitmap resources in the PNG format. The SciTech MGL can load and save all styles of PNG format bitmaps, including RGB, 8-bit grayscale and 8-bit color bitmaps with an alpha channel. The PNG bitmap format is the only format that the SciTech MGL supports that can load and handle alpha channel data. The bitmap data will be converted to the format of the destination device context if the bitmap is loaded into a device context that is of a different pixel format to the incoming bitmap.

MGL_availablePNG	MGL_getPNGSize
MGL_getPNGSizeExt	MGL_loadPNG
MGL_loadPNGExt	MGL_loadPNGIntoDC
MGL_loadPNGIntoDCExt	MGL_savePNGFromDC
MGL_savePNGFromDCExt	

Random number generation routines

The following functions provide support for generating *fast* random numbers with either 16 or 32 bits of range.

MGL_random	MGL_randoml
MGL_srand	

Rectangle and Point manipulation

The following functions provide support for performing Boolean operations on simple rectangles. Note that some operations are not exact, and sometimes the result cannot be properly represented as a single rectangle. If you need to obtain the proper result, you will need to use the complex region routines.

MGL_defRect	MGL_defRectPt
MGL_disjointRect	MGL_emptyRect
MGL_equalPoint	MGL_equalRect
MGL_insetRect	MGL_offsetRect
MGL_ptInRect	MGL_ptInRectCoord
MGL_ptInRegion	MGL_sectRect
MGL_sectRect	MGL_sectRectCoord
MGL_sectRectCoord	MGL_sectRectFast
MGL_sectRectFastCoord	MGL_unionRect
MGL_unionRect	MGL_unionRectCoord
MGL_unionRectCoord	

Window manager functions

The following functions provide support for a minimal window manager implementation in the SciTech MGL. This code is presently used to provide the low-level windowing functions that are used by the wxWindows port to the SciTech MGL.

MGL_wmBeginPaint	MGL_wmCaptureEvents
MGL_wmCoordGlobalToLocal	MGL_wmCoordLocalToGlobal
MGL_wmCreate	MGL_wmCreateWindow
MGL_wmDestroy	MGL_wmDestroyWindow
MGL_wmEndPaint	MGL_wmGetRootWindow
MGL_wmGetWindowAtPosition	MGL_wmGetWindowFlags
MGL_wmGetWindowParent	MGL_wmGetWindowUserData
MGL_wmInvalidateRect	MGL_wmInvalidateRegion
MGL_wmInvalidateWindow	MGL_wmInvalidateWindowRect
MGL_wmInvalidateWindowRegion	MGL_wmLowerWindow
MGL_wmPopGlobalEventHandler	MGL_wmPopWindowEventHandler
MGL_wmProcessEvent	MGL_wmPushGlobalEventHandler
MGL_wmPushWindowEventHandler	MGL_wmRaiseWindow
MGL_wmRemoveGlobalEventHandler	MGL_wmRemoveWindowEventHandler
MGL_wmReparentWindow	MGL_wmSetGlobalCursor
MGL_wmSetWindowCursor	MGL_wmSetWindowDestructor
MGL_wmSetWindowFlags	MGL_wmSetWindowPainter
MGL_wmSetWindowPosition	MGL_wmSetWindowUserData
MGL_wmShowWindow	MGL_wmUncaptureEvents
MGL_wmUpdateDC	

OpenGL binding functions

The SciTech MGL includes complete hardware and software support for OpenGL rendering in full-screen and windowed graphics modes. In order to use the OpenGL rendering functions, you must first create an OpenGL rendering context using the SciTech MGL OpenGL binding functions defined below. The functions are equivalent to using the wgl functions on Windows or the glx functions for X Windows development with OpenGL.

MGL_glChooseVisual	MGL_glCreateContext
MGL_glDeleteContext	MGL_glDisableMGLFuncs
MGL_glEnableMGLFuncs	MGL_glEnumerateDrivers
MGL_glGetProcAddress	MGL_glGetVisual
MGL_glHaveHWOOpenGL	MGL_glMakeCurrent
MGL_glRealizePalette	MGL_glResizeBuffers
MGL_glSetDriver	MGL_glSetOpenGLType
MGL_glSetPalette	MGL_glSetVisual
MGL_glSwapBuffers	

Event handling

The following functions provide a set of low-level routines to maintain a queue of keyboard, mouse and joystick events. Naturally the events are not restricted to just keyboard, mouse and joystick events, but can be any type of user defined event. This allows the application to easily combine the keyboard and mouse into a single system for interacting with the user.

EVT_allowLEDS	EVT_flush
EVT_getCodePage	EVT_getHeartBeatCallback
EVT_getMousePos	EVT_getNext
EVT_halt	EVT_isKeyDown
EVT_joyIsPresent	EVT_joySetCenter
EVT_joySetLowerRight	EVT_joySetUpperLeft
EVT_peekNext	EVT_pollJoystick
EVT_post	EVT_setCodePage
EVT_setHeartBeatCallback	EVT_setMousePos
EVT_setUserEventFilter	

Mouse handling

The following functions allow the application to fully control the mouse cursor such as showing and hiding the cursor and changing it's shape and color.

MS_getPos	MS_hide
MS_moveTo	MS_obscure
MS_setCursor	MS_setCursorColor
MS_setCursorColorExt	MS_show

Game Framework

The Game Framework abstracts most of the code overhead required to write full-screen and windowed applications. Using the Game Framework frees you from the tedium of writing windows classes and allows you to concentrate on what's really important, your game. The Game Framework library consists of the following functions:

GM_chooseMode	GM_cleanup
GM_exit	GM_findMode
GM_getDoDraw	GM_getExitMainLoop
GM_getHaveWin95	GM_getHaveWinNT
GM_init	GM_initPath
GM_initSysPalNoStatic	GM_initWindowPos
GM_mainLoop	GM_processEvents
GM_realizePalette	GM_setAppActivate
GM_setDrawFunc	GM_setDriverOptions
GM_setEventFunc	GM_setExitFunc
GM_setGameLogicFunc	GM_setKeyDownFunc
GM_setKeyRepeatFunc	GM_setKeyUpFunc
GM_setLeftBuffer	GM_setMode
GM_setModeExt	GM_setModeFilterFunc
GM_setModeSwitchFunc	GM_setMouseDownFunc
GM_setMouseMoveFunc	GM_setMouseUpFunc
GM_setPalette	GM_setPreModeSwitchFunc
GM_setRightBuffer	GM_setSuspendAppCallback
GM_startOpenGL	GM_startStereo
GM_stopStereo	GM_swapBuffers
GM_swapDirtyBuffers	

Sprite Manager

The Sprite Manager provides a seamless interface for using off-screen video memory to store sprites and other bitmaps for extremely fast sprite animation. The Sprite Manager layers on top of the lightweight buffer support in the SciTech MGL. The Sprite Manager library is comprised of these functions:

SPR_destroyBitmap	SPR_draw
SPR_drawExt	SPR_drawSection
SPR_drawSectionExt	SPR_mgrAddOpaqueBitmap
SPR_mgrAddTransparentBitmap	SPR_mgrEmpty
SPR_mgrExit	SPR_mgrInit

Platform Specific: Windows

The following functions are specific to the Windows version of the SciTech MGL. The SciTech MGL can run in the Windows environment as either a windowed application or as a full screen application. The API for full screen applications is identical to that for all other full screen environments. However when running in a windowed environment, you will need to write Windows specific code and use the SciTech MGL windows specific binding functions.

MGL_beginPaint	MGL_endPaint
MGL_getFullScreenWindow	MGL_registerEventProc
MGL_registerFullScreenWindow	

MGL Library Reference

This section contains a detailed reference for the SciTech MGL Graphics Library functions and data structures.

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 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_MOUSEDOWN 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 and event of the specified type is recieved.

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 functions halts exceution until an event of the specified type is recieved 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

GM_chooseMode

Display a dialog box to choose a fullscreen graphics mode

Declaration

```
ibool MGLAPI GM_chooseMode(
    GM_modeInfo *mode,
    ibool *startWindowed)
```

Prototype In

gm/gm.h

Parameters

<i>mode</i>	Place to return the selected mode information
<i>startWindowed</i>	True if use wishes to start windowed, false if not

Return Value

True if a mode was chosen, false on error or if the user clicked cancel.

Description

This function will bring up a dialog box allowing the user to interactively choose the graphics mode to be used for fullscreen modes, as well as allowing then to change the OpenGL implementation, WinDirect and DirectDraw support and also force the game to start in a window or fullscreen. This is mostly a convenience function which is great for debugging, testing and demonstration purposes. Note that if you do call this function, you must add the resources for the dialog box used to your application, which are located in the SCITECH\INCLUDE\GM\GMDLG.RC resource file.

GM_cleanup

Cleans up the Game Framework and restores original mode

Declaration

```
void MGLAPI GM_cleanup(void)
```

Prototype In

gm/gm.h

Description

This function calls the users registered exit function, exits the MGL (which puts the system back into text mode for DOS or GDI mode for Windows) and then does some final Game Framework cleanup. Normally you wont call this function unless you have replaced the *GM_mainLoop* function with your own custom version.

See Also

GM_mainLoop, *GM_processEvents*, *GM_processEventsWin*

GM_exit

Tells the Game Framework to exit the main loop

Declaration

```
void MGLAPI GM_exit(void)
```

Prototype In

gm/gm.h

Description

This function simple lets the Game Framework know that the user wants to exit and to clean up and exit from the main loop. After a call to this function, the Game Framework will return from *GM_mainLoop* when it begins to process the next frame.

See Also

GM_mainLoop

GM_findMode

Finds an available mode that has the desired resolution and color depth.

Declaration

```
ibool MGLAPI GM_findMode(
    GM_modeInfo *mode,
    int xRes,
    int yRes,
    int bits)
```

Prototype In

gm/gm.h

Parameters

<i>mode</i>	Place to store the returned graphics mode information
<i>xRes</i>	X resolution of the mode to find
<i>yRes</i>	Y resolution of the mode to find
<i>bits</i>	Color depth of the mode to find (-1 for don't care)

Return Value

True if a valid mode was found, false if not found.

Description

This function searches the list of available graphics modes for one that matches the desired resolution and color depth. This is most useful for finding a good default graphics mode to start your game in if the user has not selected a default mode yet. Note that this function searches for the mode from the top of the list backwards, so that we find the highest performance 320x200 and 320x240 modes (i.e.: the Linear Framebuffer modes rather than the VGA ModeX or Standard VGA modes).

GM_getDoDraw

Returns true if drawing is allowed

Declaration

```
ibool MGLAPI GM_getDoDraw(void)
```

Prototype In

gm/gm.h

Return Value

True if drawing is allowed

Description

This function returns the value of the global variable `GM_doDraw`, and is primarily intended as an interface function in the DLL version of the Game Framework used for Borland Delphi.

GM_getExitMainLoop

Returns true if the main loop should exit

Declaration

```
ibool MGLAPI GM_getExitMainLoop(void)
```

Prototype In

gm/gm.h

Return Value

True if the main loop should exit

Description

This function returns the value of the global variable GM_exitMainLoop, and is primarily intended as an interface function in the DLL version of the Game Framework used for Borland Delphi.

GM_getHaveWin95

Returns true if we are running on Windows 95/98/Me

Declaration

```
ibool MGLAPI GM_getHaveWin95(void)
```

Prototype In

gm/gm.h

Return Value

True when running on Windows 95/98/Me

Description

This function returns the value of the global variable GM_haveWin95, and is primarily intended as an interface function in the DLL version of the Game Framework used for Borland Delphi.

GM_getHaveWinNT

Returns true if we are running on Windows NT

Declaration

```
ibool MGLAPI GM_getHaveWinNT(void)
```

Prototype In

gm/gm.h

Return Value

True when running on Windows NT

Description

This function returns the value of the global variable GM_haveWinNT, and is primarily intended as an interface function in the DLL version of the Game Framework used for Borland Delphi.

GM_init

initializes the Game Framework

Declaration

```
GMDC * MGLAPI GM_init(
    const char *windowTitle)
```

Prototype In

gm/gm.h

Parameters

windowTitle Title for window in windowed modes and on task bar

Return Value

Pointer to the game framework context object

Description

This function initializes the Game Framework and must be called before you attempt to set a graphics mode. Once this function has been called, the Game Framework will have enumerated all the available graphics modes and stored this information into the `modeList` field of the `GMDC` structure returned from this function. It is then up to the application to find a suitable mode and initialized it with a call to `GM_setMode`.

Before you can do anything useable with the Game Framework, after you have called the `GM_init` function, you must then register a number of function callbacks with the Game Framework that it will call to implement the 'body' of the game (similar to C++ virtual functions, but in C). Two of the most important are `GM_setDrawFunc` and `GM_setGameLogicFunc`. If you want to respond to keyboard commands you will probably also want to call `GM_setKeyDownFunc` as well.

Note: *The Game Framework is responsible for creating the main window used by the game. Hence the value you pass in for `windowTitle` will be the main title for your games window in windows modes, as well as the title that the user will see when your game is minimised to the task bar in Windows 95 and Windows NT 4.0.*

Note: *The Game Framework only creates and maintains a single window for the life of the game, and on switches between windowed and fullscreen modes will automatically change the attributes of the main window for the appropriate mode. This way your game only needs to register a single main window with DirectSound and other DirectX components during initialization time, and avoids the problems of re-starting DirectSound during mode switches (and hence you sound can continue to play as you switch on the fly between resolutions and fullscreen and windowed modes).*

See Also

`GM_setDriverOptions`, `GM_setMode`, `GM_setDrawFunc`, `GM_setGameLogicFunc`

GM_initPath

Tells the Game Framework where to find the MGL resources

Declaration

```
void MGLAPI GM_initPath(  
    const char *MGLPath)
```

Prototype In

gm/gm.h

Parameters

MGLPath path to MGL resources

Description

This function tells the Game Framework where to find the MGL resources such as bitmaps, fonts etc. By default the MGL always looks in the current directory, however you can use this to point to a directory that contains the resources such as on a CD-ROM drive. Functions such as *MGL_loadBitmap* and *MGL_loadFont* will use this path to locate the files.

See Also

GM_init

GM_initSysPalNoStatic

Disables support for static system palette colors

Declaration

```
void MGLAPI GM_initSysPalNoStatic(  
    ibool flag)
```

Prototype In

gm/gm.h

Parameters

flag True to disable static system colors

Description

This function is used to inform the Game Framework whether you wish to bypass the static system palette in your game and use all available colors (except 0 and 255 which are always black and white respectively). By default the static colors are left alone and the MGL will fix up the hardware palette to use the proper static colors (by default this value is set to false and the static system colors cannot be changed).

See Also

GM_init

GM_initWindowPos

Sets the default window position for windowed modes

Declaration

```
void MGLAPI GM_initWindowPos (  
    int x,  
    int y)
```

Prototype In

gm/gm.h

Parameters

<i>x</i>	X coordinate for top left corner of window
<i>y</i>	Y coordinate for top left corner of window

Description

This function tells the Game Framework where you want the top left corner of the window to be positioned for windowed modes. By default the Game Framework will center the window on the screen the first time the window is created, and you can use this function to override the default behavior.

See Also

GM_init

GM_mainLoop

Runs the main loop for the Game Framework

Declaration

```
void MGLAPI GM_mainLoop(void)
```

Prototype In

gm/gm.h

Description

This function is the main event loop for the Game Framework, and controls execution of the game until the game exists. This function is responsible for farming out events to the event handling callbacks registered by the game along with calling the game's gameLogic and draw callbacks. Note that if we are suspended on the task bar we continue to process events and call the gameLogic function (so networking can continue) however we skip the call the to drawing function to avoid attempting to write to video memory we no longer own.

The main loop is a simple loop constructed of the following steps (note that MyGameLogic and MyDrawFrame are assumed to be your game logic and draw functions that you would normally register with the Game Framework before calling the GM_mainLoop function):

```
GM_exitMainLoop = false;
while (!GM_exitMainLoop) {
    GM_processEvents();
    MyGameLogic();
    if (GM_doDraw)
        MyDrawFrame();
}
GM_cleanup();
```

If you wish to replace the main loop with your own, you can take the existing main loop code and replace it with your own variations. The GM_processEvents functions processes events via the MGL event handling functions and dispatches them to the registered event callbacks. If you wish you can call GM_processEventsWin instead, which will simply flush the windows message queue and you will be expected to handle all keyboard and mouse events with the window procedure registered with GM_registerEventProc. Hence an alternate main loop with all event handling done in regular window procedure would be coded as follows:

```
GM_exitMainLoop = false;
GM_registerEventProc(MyWindowProc);
while (!GM_exitMainLoop) {
    GM_processEventsWin();
    MyGameLogic();
    if (GM_doDraw)
        MyDrawFrame();
}
```

```
    }  
    GM_cleanup();
```

Note: *If you do replace the main loop with your own, make absolutely sure that you don't call your draw function if the global variable GM_doDraw is set to false, otherwise your game could lock the system when the user Alt-Tabs away and back to the desktop. Also make sure that you call GM_cleanup on the way out.*

Note: *To exit the main loop, call the GM_exit function which lets the main loop know that it should exit on the next iteration.*

Note: *You must call GM_processEvents or GM_processEventsWin in your main loop to ensure that the Game Framework has a chance to process some internal functions every loop.*

See Also

GM_init, GM_setDrawFunc, GM_setGameLogicFunc, GM_exit, GM_processEvents,
GM_processEventsWin, GM_cleanup

GM_processEvents

Processes all events for the current iteration of the main loop

Declaration

```
void MGLAPI GM_processEvents(void)
```

Prototype In

gm/gm.h

Description

This function is the event processing handler for the main event loop for the Game Framework. This function is responsible for farming out events to the event handling callbacks registered by the game.

Note: *If you wish to process messages in your game using a regular Windows window procedure, replace the GM_mainLoop function with your own and call GM_processEventsWin instead.*

See Also

GM_mainLoop, GM_processEventsWin, GM_cleanup

GM_processEventsWin

Processes all Windows events for the current iteration of the main loop

Declaration

```
void MGLAPI GM_processEventsWin(void)
```

Prototype In

gm/gm.h

Description

This function is the message processing handler for the main event loop for the Game Framework. This function basically processes all windows messages and passes them to the window procedure for handling. Note that this function does not use the MGL's event handling routines, and is specific to Windows. Essentially this function implements the following:

```
while (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE)) {  
    TranslateMessage(&msg);  
    DispatchMessage(&msg);  
}
```

See Also

GM_mainLoop, GM_processEvents, GM_cleanup

GM_realizePalette

Programs the hardware color palette

Declaration

```
void MGLAPI GM_realizePalette(
    int numColors,
    int startIndex,
    int waitVRT)
```

Prototype In

gm/gm.h

Parameters

<i>numColors</i>	Number of colors to set
<i>startIndex</i>	Starting index in device context palette
<i>waitVRT</i>	True to wait for vertical retrace

Description

This function programs the hardware color palette from the current display device context. You should first call *GM_setPalette* to set the color palette entries to the values that you require before calling this function.

Note: *If we have a memory back buffer we also realize the palette values for this to ensure we have an identity palette for blit operations for maximum speed.*

See Also

GM_setPalette

GM_registerEventProc

Registers a user event procedure with the Game Framework

Declaration

```
void MGLAPI GM_registerEventProc(  
    MGL_WNDPROC winproc)
```

Prototype In

gm/gm.h

Parameters

winproc User window procedure to register

Description

This function registers a user window procedure with the Game Framework. The primary purpose of this function is to allow your game to process regular windows messages (such as CD-Audio notification messages and other messages not automatically handled by the Game Framework) in your game. To use this function, simply write a regular window procedure as you would normally for a real window, but instead of registering the window procedure with in the RegisterClass function (the Game Framework calls RegisterClass for you during initialization) call this function to register your window procedure with the Game Framework.

Note: *This function is only available in the Windows version of the Game Framework.*

GM_registerMainWindow

Registers a user main window with the Game Framework

Declaration

```
void MGLAPI GM_registerMainWindow(MGL_HWND
    hwndMain)
```

Prototype In

gm/gm.h

Parameters

hwndMain Handle to the main window for the application

Description

This function registers a user main window with the Game Framework. The primary purpose of this function is to allow your main game code to do the creation of the main window that is used by the Game Framework, instead of letting the Game Framework libraries do it for you. This is mostly to support integrating the Game Framework code with existing game code that already does window creation and message handling.

Note that you should only call this function *after* you have called *GM_init* to initialize the Game Framework.

Note: *If you use this function to register a main window, do not use the GM_registerEventProc function to register your window procedure with the Game Framework!!*

Note: *This function is only available in the Windows version of the Game Framework.*

GM_setAppActivate

Sets the application activate callback function

Declaration

```
void MGLAPI GM_setAppActivate(
    GM_activateFunc func)
typedef void (*GM_activateFunc) (
    ibool active)
```

Prototype In

gm/gm.h

Parameters

func Application activate callback function to register

Description

This function sets the application activate callback function for your Game Framework game. This function is called in windowed modes whenever the activation status of your game changes, which can occur if the window is minimised to the task bar or if the user switches away to another application using *Alt-Tab*. Your callback is passed a flag that indicates whether your game is now currently active or not, and should be used to enable and disable support for things such as CD-Audio when your application loses activation (or the current focus).

Note: *The Game Framework contains built in support for enabling and disabling the static system palette colors when running in windowed modes, and will automatically switch back to static system color mode when your window loses the activation focus. Hence you should not attempt to change this in your application activate callback (if you want to be able to use the static system colors in a window, call GM_initSysPalNoStatic(true) before you initialize the Game Framework.*

Note: *This function is only called in windowed modes, and the equivalent function for fullscreen modes is set with the GM_setSuspendAppCallback function.*

See Also

GM_init, GM_setSuspendAppCallback, GM_initSysPalNoStatic

GM_setDrawFunc

Sets the draw callback function

Declaration

```
void MGLAPI GM_setDrawFunc (
    GM_drawFunc func)
typedef void (*GM_drawFunc) (void)
```

Prototype In

gm/gm.h

Parameters

func Draw callback function to register

Description

This function sets the draw callback function for your Game Framework game and is called by the Game Framework main loop (*GM_mainLoop*) once per frame to draw the current frame in the game. Note that the Game Framework expects this function both draw the current frame and swap the buffers to make them visible using *GM_swapBuffers* or *GM_swapDirtyBuffers*.

Note: *In order to be able to continue running your games main logic loops while the user has switch away (ie: Alt-Tab) from your game in Windows, while the application is minimised we continue to process messages and call the registered game logic callback, however the draw callback will not be called until the application is restored to fullscreen mode. Hence your draw callback should not contain any game logic functionality, but only contain code to draw the current frame in the game.*

See Also

GM_init, *GM_swapBuffers*, *GM_swapDirtyBuffers*

GM_setDriverOptions

Sets driver registration options for the Game Framework

Declaration

```
void MGLAPI GM_setDriverOptions(  
    GM_driverOptions *opt)
```

Prototype In

gm/gm.h

Parameters

opt Parameter block containing driver registration options

Description

This function tells the Game Framework which driver technologies you want to support in your game. By default all of them are enabled, and you can use this function to disable certain driver technologies at runtime for compatibility in the field. You may call this function as many times as you wish to change the driver options on the fly, and if the values change the Game Framework will re-enumerate the list of available graphics modes for you.

Note that the *GM_driverOptions* structure also contains the *modeFlags* field which represents the color depths that you will be supporting in your application, so that the Game Framework will only enumerate modes that your game can support. For instance if you only support 8bpp modes, then pass a value of *GM_MODE_8BPP*. If you support 8bpp and 15/16bpp then pass in a value of *GM_MODE_8BPP | GM_MODE_16BPP*. Note also that you can change the supported mode flags at any time, which is useful if your software renderer only supports 8bpp modes, while in 3D hardware accelerated modes you want to support all available color depths.

Note: *The Game Framework enumerates both 15bpp (5:5:5) and 16bpp (5:6:5) modes when you pass in a value of GM_MODE_16BPP, since both of these modes will likely be available on end user systems. Also note that it is equally likely that in some cases only one of these formats and not the other is supported on the end user system, so your code will have to be able to support both formats for compatibility.*

Note: *If your game does not require hardware OpenGL support (ie: you are not using OpenGL in your game), then you should set the useOpenGLHw flag to false to make sure that the OpenGL drivers are not registered with the MGL (which would require the OpenGL runtime DLL's to be installed on your end users system).*

Note: *We recommend that you provide support for disabling both DirectDraw and WinDirect modes via a command line switch in your game, in case either of these two technologies have problems on your customer machines. Please see the Game Framework sample code for ideas on how to do this.*

See Also

GM_init

GM_setEventFunc

Sets the event callback function

Declaration

```
void MGLAPI GM_setEventFunc(  
    GM_eventFunc func)  
typedef void (*GM_eventFunc) (  
    event_t *evt)
```

Prototype In

gm/gm.h

Parameters

func Event callback function to register

Description

This function sets the general event callback function for your Game Framework game and is called by the Game Framework main loop (*GM_mainLoop*) multiple times per frame to process all non key and non mouse related events in the order that they are entered by the user (ie: timer or user events).

GM_setExitFunc

Sets the exit callback function

Declaration

```
void MGLAPI GM_setExitFunc(  
    GM_exitFunc func)  
typedef void (*GM_exitFunc)(void)
```

Prototype In

gm/gm.h

Parameters

func Exit callback function to register

Description

This function sets the exit callback function for your Game Framework game and is called by the Game Framework main loop just before calling *MGL_exit* to shut down the MGL and return to windowed mode before returning to your code. If you have any code that must be called before exiting from fullscreen mode, you should register it with this function.

Note: *This function will always be called after the mode switch has occurred and the system is in the new graphics mode.*

See Also

GM_init, *GM_mainLoop*

GM_setGameLogicFunc

Sets the game logic callback function

Declaration

```
void MGLAPI GM_setGameLogicFunc(
    GM_gameFunc func)
typedef void (*GM_gameFunc) (void)
```

Prototype In

gm/gm.h

Parameters

func Game logic callback function to register

Description

This function sets the game logic callback function for your Game Framework game and is called by the Game Framework main loop (*GM_mainLoop*) once per frame to update the game logic for the next frame after drawing the current frame.

Note: *In order to be able to continue running your games main logic loops while the user has switch away (ie: Alt-Tab) from your game in Windows, while the application is minimised we continue to process messages and call the registered game logic callback, however the draw callback will not be called until the application is restored to fullscreen mode. Hence your game logic callback should not contain any code that performs drawing to the screen, as all that code should be located in your draw callback function.*

See Also

GM_init, *GM_setDrawFunc*

GM_setKeyDownFunc

Sets the key down callback function

Declaration

```
void MGLAPI GM_setKeyDownFunc(
    GM_keyDownFunc func)
typedef void (*GM_keyDownFunc) (
    event_t *evt)
```

Prototype In

gm/gm.h

Parameters

func Key down callback function to register

Description

This function sets the key down callback function for your Game Framework game and is called by the Game Framework main loop (*GM_mainLoop*) multiple times per frame to process all key down events in the order that they are entered by the user. Note that your callback is passed a copy of the event that needs to be processed, so you can if you wish have a single handler for all key events, or one that handles both key down and key repeat events and register the same function with the Game Framework for both event types.

Note: *The key down callback will not be called for key repeat events (ie: the user holds a key down). If you wish to capture key repeat events, use GM_setKeyRepeatFunc.*

See Also

GM_init, GM_setKeyRepeatFunc, GM_setKeyUpFunc

GM_setKeyRepeatFunc

Sets the key repeat callback function

Declaration

```
void MGLAPI GM_setKeyRepeatFunc (
    GM_keyRepeatFunc func)
typedef void (*GM_keyRepeatFunc) (
    event_t *evt)
```

Prototype In

gm/gm.h

Parameters

func Key repeat callback function to register

Description

This function sets the key repeat callback function for your Game Framework game and is called by the Game Framework main loop (*GM_mainLoop*) multiple times per frame to process all key repeat events in the order that they are entered by the user. Note that your callback is passed a copy of the event that needs to be processed, so you can if you wish have a single handler for all key events, or one that handles both key down and key repeat events and register the same function with the Game Framework for both event types.

See Also

GM_setKeyDownFunc, *GM_setKeyUpFunc*

GM_setKeyUpFunc

Sets the key up callback function

Declaration

```
void MGLAPI GM_setKeyUpFunc (  
    GM_keyUpFunc func)  
typedef void (*GM_keyUpFunc) (  
    event_t *evt)
```

Prototype In

gm/gm.h

Parameters

func Key up callback function to register

Description

This function sets the key up callback function for your Game Framework game and is called by the Game Framework main loop (*GM_mainLoop*) multiple times per frame to process all key up events in the order that they are entered by the user. Note that your callback is passed a copy of the event that needs to be processed, so you can if you wish have a single handler for all key events if you wish.

See Also

GM_setKeyDownFunc, *GM_setKeyRepeatFunc*

GM_setLeftBuffer

Set active rendering to the left stereo buffer.

Declaration

```
void MGLAPI GM_setLeftBuffer(void)
```

Prototype In

gm/gm.h

Description

This function sets the active rendering buffer for all subsequent drawing commands to the left stereo buffer.

See Also

GM_setRightBuffer

GM_setMode

Switches to a fullscreen or windowed graphics mode

Declaration

```
ibool MGLAPI GM_setMode(
    GM_modeInfo *info,
    ibool windowed,
    int pages,
    ibool forceSysMem)
```

Prototype In

gm/gm.h

Parameters

<i>info</i>	Structure describing the mode to initialize
<i>windowed</i>	True if the mode should be in a window
<i>pages</i>	Number of pages for hardware buffering
<i>forceSysMem</i>	Flag to force a system memory buffer for all drawing

Return Value

True on success, false on error

Description

This function is now obsolete, and has been replaced by the *GM_setModeExt* function. You should use the new function for all new application development.

See Also

GM_init, *GM_setModeSwitchFunc*, *GM_setModeExt*

GM_setModeExt

Switches to a fullscreen or windowed graphics mode

Declaration

```
ibool MGLAPI GM_setModeExt (
    GM_modeInfo *info,
    ibool windowed,
    int pages,
    int refreshRate,
    ibool forceSysMem,
    ibool stereo)
```

Prototype In

gm/gm.h

Parameters

<i>info</i>	Structure describing the mode to initialize
<i>windowed</i>	True if the mode should be in a window
<i>pages</i>	Number of pages for hardware buffering
<i>refreshRate</i>	Desired refresh rate for fullscreen display mode
<i>forceSysMem</i>	Flag to force a system memory buffer for all drawing
<i>stereo</i>	True to enable stereo rendering

Return Value

True on success, false on error

Description

This function sets a graphics mode for the game given the passed in mode information. If the windowed parameter is set to true, a windowed mode will be set otherwise a fullscreen graphics mode will be set. You must pass in one of the modes listed in the *GMDC* modeList returned from *GM_init* to this function. When the Game Framework switches to the windowed equivalent to a fullscreen graphics mode, we search for the closest 1:1 aspect ratio window size for the mode that corresponds to the fullscreen mode chosen. Hence for a 320x200 fullscreen mode we choose a 320x240 window size, for 320x400 and 320x480 we choose a 320x240 window size and for 640x350 and 640x400 we choose a 640x480 window size. This way we won't end up with a window on the desktop that is not a 1:1 aspect ratio and desktop modes are generally 1:1 aspect ratio.

The number of pages passed in is used to allocate the specified number of hardware video memory pages for multi-buffering and may be any value. The Game Framework will however lower this value to the maximum that the hardware supports, and if only 1 hardware page is available a system memory back buffer will be created automatically. Hence if you want to always try and use triple buffering if available, pass a value of 3 when you call this function.

The refresh rate parameter is used to select a desired refresh rate for the fullscreen display mode. If that refresh rate cannot be achieved due to hardware constraints, the

next lowest available refresh rate will be used. If you wish to use the user default setting for the refresh rate (always a good idea unless you need refresh control, such as for stereo), simply pass in a value of MGL_DEFAULT_REFRESH.

The stereo flag is used to enable support for either software or hardware stereo modes in the MGL. This will fail if the underlying hardware device does not support stereo capabilities. Stereo can be used with OpenGL using Mesa, SGI OpenGL, SciTech DirectGL or Microsoft OpenGL (Microsoft OpenGL requires a stereo ready OpenGL driver to be installed).

Note also that the Game Framework automatically handles switching between windowed and fullscreen modes on the fly. By default the Game Framework contains code to provide two methods of switching to fullscreen modes when running in windowed modes:

- 1 When the user hits the *Alt-Enter* key combination
- 2 When the user clicks the *Maximise* button on the games title bar

Likewise when the game is running in a fullscreen mode and the user hits the *Alt-Enter* key, the video mode will automatically be switched to windowed mode. Unless you have registered a mode switch callback with *GM_setModeSwitchFunc*, on the fly switching between fullscreen and windowed modes is disabled.

Note: *You may call this function while already in a windowed or fullscreen graphics mode, which will cause the current graphics mode to be changed on the fly.*

Note: *If you wish to force the Game Framework to always create a system memory buffer for rendering, regardless of how many display pages are available, set the *forceSysMem* field to true when calling this function. Note also that if you have requested multiple buffers, all those buffers will still be used so you will see no tearing, however all rendering will be performed to a system buffer which will be copied to the screen when you swap the buffers with *GM_swapBuffers*.*

Drawing directly to a hardware linear framebuffer is usually extremely fast and will provide the most efficient method of rendering. However if your rendering requires reads from the framebuffer (for effects such as blending), this will be very slow over the PCI bus and your code will be faster if you force a system member back buffer.

Note: *If the user dynamically switches the resolution or color depth of the windows desktop while your application is running, if you have registered a mode switch callback with the Game Framework (via *GM_setModeSwitchFunc*) then the Game Framework will switch to a new windowed mode to cause the memory back buffers to be re-allocated for the most optimal format for the Windows display mode.*

See Also

GM_init, *GM_setModeSwitchFunc*

GM_setModeFilterFunc

Sets the custom mode filter for mode enumeration

Declaration

```
void MGLAPI GM_setModeFilterFunc(  
    GM_modeFilterFunc filter)  
typedef ibool (*GM_modeFilterFunc) (  
    int xRes,  
    int yRes,  
    int bits,  
    ulong flags)
```

Prototype In

gm/gm.h

Parameters

filter New mode filter to set

Description

This function allows you to register a mode filter callback with the Game Framework, which will be called during mode enumeration and will allow you to apply your own custom filtering code to the list of available video modes. Hence you can use this function to filter out all non 1:1 aspect ratio modes for instance.

See Also

GM_init

GM_setModeSwitchFunc

Sets the mode switch callback function

Declaration

```
void MGLAPI GM_setModeSwitchFunc(
    GM_modeSwitchFunc func)
typedef void (*GM_modeSwitchFunc)(
    GM_modeInfo *mode,
    ibool windowed)
```

Prototype In

gm/gm.h

Parameters

func Mode switch callback function to register

Description

This function sets the mode switch callback function for your Game Framework game and is called by the Game Framework when automatically switching on the fly between windowed and fullscreen modes. By default this handler is set to NULL, and unless you call this function support for switching on the fly between windowed and fullscreen modes is disabled. By default the Game Framework contains code to provide two methods of switching to fullscreen modes when running in windowed modes:

- 1 When the user hits the *Alt-Enter* key combination
- 2 When the user clicks the *Maximise* button on the games title bar

Likewise when the game is running in a fullscreen mode and the user hits the *Alt-Enter* key, the video mode will automatically be switched to windowed mode. In order to support auto-switching between fullscreen and windowed modes, all the MGL device contexts will be destroyed and re-created during the switch, so you will have to include other code to re-initialize the MGL to the state that the game is currently in (ie: setting the color palette etc.) in your mode switch callback. You will also need to code your game in such as way that it can handle dynamic resolution changes on the fly.

See Also

GM_init, *GM_setMode*

GM_setMouseDownFunc

Sets the mouse down callback function

Declaration

```
void MGLAPI GM_setMouseDownFunc (
    GM_mouseDownFunc func)
typedef void (*GM_mouseDownFunc) (
    event_t *evt)
```

Prototype In

gm/gm.h

Parameters

func Mouse down callback function to register

Description

This function sets the mouse down callback function for your Game Framework game and is called by the Game Framework main loop (*GM_mainLoop*) multiple times per frame to process all mouse down events in the order that they are entered by the user. Note that your callback is passed a copy of the event that needs to be processed, so you can if you wish have a single handler for all mouse events, or one that handles both mouse down and mouse up events and register the same function with the Game Framework for both event types.

See Also

GM_init, *GM_setMouseUpFunc*, *GM_setMouseMoveFunc*

GM_setMouseMoveFunc

Sets the mouse move callback function

Declaration

```
void MGLAPI GM_setMouseMoveFunc(  
    GM_mouseMoveFunc func)  
typedef void (*GM_mouseMoveFunc) (  
    event_t *evt)
```

Prototype In

gm/gm.h

Parameters

func Mouse move callback function to register

Description

This function sets the mouse move callback function for your Game Framework game and is called by the Game Framework main loop (*GM_mainLoop*) multiple times per frame to process all mouse move events in the order that they are entered by the user.

See Also

GM_init, *GM_setMouseDownFunc*, *GM_setMouseUpFunc*

GM_setMouseUpFunc

Sets the mouse up callback function

Declaration

```
void MGLAPI GM_setMouseUpFunc(
    GM_mouseUpFunc func)
typedef void (*GM_mouseUpFunc) (
    event_t *evt)
```

Prototype In

gm/gm.h

Parameters

func Mouse up callback function to register

Description

This function sets the mouse up callback function for your Game Framework game and is called by the Game Framework main loop (*GM_mainLoop*) multiple times per frame to process all mouse up events in the order that they are entered by the user. Note that your callback is passed a copy of the event that needs to be processed, so you can if you wish have a single handler for all mouse events, or one that handles both mouse down and mouse up events and register the same function with the Game Framework for both event types.

See Also

GM_init, *GM_setMouseDownFunc*, *GM_setMouseMoveFunc*

GM_setPalette

Sets the color palette entries

Declaration

```
void MGLAPI GM_setPalette(
    palette_t *pal,
    int numColors,
    int startIndex)
```

Prototype In

gm/gm.h

Parameters

<i>pal</i>	Array of palette values to set
<i>numColors</i>	Number of colors to set
<i>startIndex</i>	Starting index in device context palette

Description

This function sets the palette values for the currently activate Game Framework device context to the values passed in pal. Note that this function does not program the hardware palette, but simply updates the internal palette values for the MGL device context. Once you have set the values in the palette, you should then call *GM_realizePalette* to program the hardware palette entries from the values currently stored in the MGL display device context.

Note: *If we have a memory back buffer we also set the palette values for this to ensure we have an identity palette for blit operations for maximum speed.*

See Also

GM_realizePalette

GM_setPreModeSwitchFunc

Sets the pre-mode switch callback function

Declaration

```
void MGLAPI GM_setPreModeSwitchFunc (
    GM_preModeSwitchFunc func)
typedef ibool (*GM_preModeSwitchFunc) (
    GM_modeInfo *mode,
    ibool windowed)
```

Prototype In

gm/gm.h

Parameters

func Pre-mode switch callback function to register

Description

This function sets the pre-mode switch callback function for your Game Framework game and is called by the Game Framework when automatically switching on the fly between windowed and fullscreen modes. If you have registered a mode switch function with *GM_setModeSwitchFunc*, your pre-mode switch function will be called before the existing mode is destroyed, giving your code a change to destroy any internal data structures that might need to be cleaned up before the mode is destroyed and the new one created.

See Also

GM_init, *GM_setMode*, *GM_setModeSwitchFunc*

GM_setRightBuffer

Set active rendering to the right stereo buffer.

Declaration

```
void MGLAPI GM_setRightBuffer(void)
```

Prototype In

gm/gm.h

Description

This function sets the active rendering buffer for all subsequent drawing commands to the right stereo buffer.

See Also

GM_setLeftBuffer

GM_setSuspendAppCallback

Sets the suspend app callback function

Declaration

```
void MGLAPI GM_setSuspendAppCallback(
    MGL_suspend_cb_t saveState)
```

Prototype In

gm/gm.h

Description

This function sets the suspend app callback function for your Game Framework game and is called by the Game Framework whenever the user switches away from your game (suspends it, such as with Alt-Tab). The Game Framework registers a default suspend application callback with the MGL to do most of the handling for you. Note however that by default the Game Framework suspend app callback passes a return value of MGL_SUSPEND_APP back to the MGL which will suspend execution of your game until it has been restored. The Game Framework has code to automatically ensure that the draw callback is not called when the game is minimised and gets re-enabled when the game is restored again if you return a value of MGL_NO_SUSPEND_APP from your registered callback (this way you can continue to run networking code in the background to keep other network players running if the server is temporarily minimised).

You should register your own version of this function to handle extra things during suspend and restores such as suspending CD-Audio sound playback or other stuff not automatically handled by the underlying multi-media libraries. A typical suspend application callback might be coded as follows:

```
int _ASMAPI SuspendAppProc(MGLDC *dc,int flags)
{
    if (flags == MGL_DEACTIVATE) {
        // Disable CD-Audio
        // Do other disabling stuff
        return MGL_NO_SUSPEND_APP;
    }
    else if (flags == MGL_REACTIVATE) {
        // Re-enable CD-Audio
        // Do other re-enabling stuff
        return MGL_NO_SUSPEND_APP;
    }
}
```

See Also

GM_init, GM_setSuspendAppProc

GM_startOpenGL

Starts OpenGL graphics for the game

Declaration

```
ibool MGLAPI GM_startOpenGL(  
    MGL_glContextFlagsType flags)
```

Prototype In

gm/gm.h

Parameters

flags OpenGL Rendering Context flags

Return Value

True on success, false on error

Description

This functions enables support for the OpenGL 3D API for the Game Framework, and after this call you must do all rendering via calls to the OpenGL API. The flags parameter (of type *MGL_glContextFlagsType*) is used to specify the type of OpenGL rendering context that you want, such as if you want RGB or color index mode, single or double buffering, an alpha buffer, an accumulation buffer, a depth buffer (z-buffer) and a stencil buffer.

If you pass in a value of *MGL_GL_VISUAL* for the flags parameter, the MGL will use the OpenGL visual that was set by a previous call to *MGL_glSetVisual*. Hence if you require more control over the type of OpenGL rendering context that is created, you can call *MGL_glChooseVisual* and *MGL_glSetVisual* before calling this function. Note that you should **not** call *MGL_glCreateContext* when using the Game Framework, but call this function instead.

Note: *After this function has been called, the current rendering context will have been made the current OpenGL rendering context with a call to *MGL_glMakeCurrent*, so you can simply start issuing OpenGL rendering commands to start drawing after calling this function.*

See Also

GM_setMode, *MGL_glChooseVisual*, *MGL_glSetVisual*

GM_startStereo

Start stereo display mode.

Declaration

```
void MGLAPI GM_startStereo(void)
```

Prototype In

gm/gm.h

Description

This function starts stereo display mode for 3D stereo viewing. Stereo display mode can be turned on and off, which is useful when displaying static screens such as menus etc.

See Also

GM_stopStereo

GM_stopStereo

Stop stereo display mode.

Declaration

```
void MGLAPI GM_stopStereo(void)
```

Prototype In

gm/gm.h

Description

This function stops stereo display mode for 3D stereo viewing. Stereo display mode can be turned on and off, which is useful when displaying static screens such as menus etc.

See Also

GM_startStereo

GM_swapBuffers

Swaps the display buffers for the game

Declaration

```
void MGLAPI GM_swapBuffers(
    MGL_waitVRTFlagType waitVRT)
```

Prototype In

gm/gm.h

Parameters

waitVRT Wait for vertical retrace flag

Description

Swaps the display buffers for the Game Framework game. If there are multiple hardware display pages enabled for the game, the waitVRT flag (of *MGL_waitVRTFlagType*) is used to determine if the MGL should wait for the vertical retrace before swapping display pages or not.

If you started the Game Framework and requested a system memory back buffer, enabled stretching or there was only one hardware display page available, the Game Framework will blit the entire system memory back buffer to the display device context and then perform the hardware page flip.

Note: *If you are intentionally using a system memory back buffer, you may want to maintain a set of dirty rectangles for your display pages and call GM_swapDirtyBuffers to swap only the dirty portions of the frames to speed things up. This is most useful for games that don't update large portions of the screen very frequently.*

See Also

GM_swapDirtyBuffers, GM_setLeftBuffer, GM_setRightBuffer

GM_swapDirtyBuffers

Swaps the display buffers for the game with dirty rectangles

Declaration

```
void MGLAPI GM_swapDirtyBuffers(
    region_t *dirty,
    MGL_waitVRTFlagType waitVRT)
```

Prototype In

gm/gm.h

Parameters

<i>dirty</i>	Region of dirty rectangles to blit
<i>waitVRT</i>	Wait for vertical retrace flag

Description

Swaps the display buffers for the Game Framework game by blitting the list of dirty rectangles to the display. The list of dirty rectangles is passed in as an MGL region, which you can construct using the MGL region manipulation functions.

If there are multiple hardware display pages enabled for the game, the waitVRT flag (of *MGL_waitVRTFlagType*) is used to determine if the MGL should wait for the vertical retrace before swapping display pages or not.

Note: *You should make sure you first call MGL_optimiseRegion before you call this function to minimise the number of rectangles in the dirty rectangle list. If you dont do this, the result will be the same but it may take longer to perform the blitting.*

Note: *If you did not specifically request a system memory back buffer, this function will behave identically to GM_swapBuffers and no blitting will occur.*

See Also

GM_swapBuffers

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 2³² microseconds, or about 1hr 20mins. For timing longer periods use the Ultra Long Period Zen Timer.*

See Also

LZTimerOffExt, LZTimerLapExt, LZTimerCountExt

MGL_FixDiv

Divides a fixed point number by another.

Declaration

```
fix32_t MGLAPI MGL_FixDiv(  
    fix32_t f,  
    fix32_t g)
```

Prototype In

mgraph.h

Return Value

Result of the division.

Description

Divides a fixed point number by another fixed point number. The idea is relatively simple; We want to set up a 64 bit dividend to be divided by our 32 bit divisor, which will give us a new 32 bit result.

See Also

MGL_FixMul, *MGL_FixMulDiv*

MGL_FixMul

Multiplies two fixed point number in 16.16 format

Declaration

```
fix32_t MGLAPI MGL_FixMul(
    fix32_t f,
    fix32_t g)
```

Prototype In

mgraph.h

Return Value

Result of the multiplication.

Description

Multiplies two fixed point number in 16.16 format together and returns the result. We cannot simply multiply the two 32 bit numbers together since we need to shift the 64 bit result right 16 bits, but the result of a FxFixed multiply is only ever 32 bits! Thus we must resort to computing it from first principles (this is slow and should ideally be re-coded in assembler for the target machine).

We can visualise the fixed point number as having two parts, a whole part and a fractional part:

$$\text{FxFixed} = (\text{whole} + \text{frac} * 2^{-16})$$

Thus if we multiply two of these numbers together we get a 64 bit result:

$$\begin{aligned} &(\text{f_whole} + \text{f_frac} * 2^{-16}) * (\text{g_whole} + \text{g_frac} * 2^{-16}) = \\ &(\text{f_whole} * \text{g_whole}) + \\ &(\text{f_whole} * \text{g_frac}) * 2^{-16} + \\ &(\text{g_whole} * \text{f_frac}) * 2^{-16} + \\ &(\text{f_frac} * \text{g_frac}) * 2^{-32} \end{aligned}$$

To convert this back to a 64 bit fixed point number to 32 bit format we simply shift it right by 16 bits (we can round it by adding 2^{-17} before doing this shift). The formula with the shift integrated is what is used below. Natrually you can alleviate most of this if the target machine can perform a native 32 by 32 bit multiplication (since it will produce a 64 bit result).

See Also

MGL_FixDiv, *MGL_FixMulDiv*

MGL_FixMulDiv

Multiplies a fixed point number by another and divides by a third number.

Declaration

```
fix32_t MGLAPI MGL_FixMulDiv(  
    fix32_t a,  
    fix32_t b,  
    fix32_t c)
```

Prototype In

mgraph.h

Parameters

<i>a</i>	First number to multiply
<i>b</i>	Second number to multiply
<i>c</i>	Third number to divide by

Return Value

Results of the multiplication and division.

Description

This function multiplies a 16.16 fixed point number by another producing a 32.32 intermediate result. This 32.32 result is then divided by another 16.16 number to produce a 16.16 result. Because this routine maintains maximum precision during the multiplication stage, you can multiply numbers that would normally overflow the standard *MGL_FixMul* function.

See Also

MGL_FixMul, *MGL_FixDiv*

MGL_addCustomMode

Adds a new custom display mode to the mode list

Declaration

```
ibool MGLAPI MGL_addCustomMode(
    int xRes,
    int yRes,
    int bitsPerPixel)
```

Prototype In

mgraph.h

Parameters

<i>xRes</i>	Horizontal resolution for the display mode in pixels
<i>yRes</i>	Vertical resolution for the display mode in lines
<i>bitsPerPixel</i>	Color depth for the display mode

Return Value

True on success, false on failure.

Description

This function attempts to use the SciTech SNAP Graphics API functions to create a new custom display mode. If the custom display mode creation succeeds, this function returns true. If it fails it returns false. The most common scenario for the failure to create a custom display mode, is that the requested mode is not compatible with the underlying hardware. In this case you may want to search for a compatible mode by stepping the X resolution in 8 pixel increments until you find a suitable mode that is larger than what you want.

Note: *Some display hardware will require that the X resolution be aligned on a 16 pixel boundary, so might fail if you try to set a mode that where the X resolution is not divisible by 16. Hence we advise that when you create new modes you try to use modes that are divisible by 16 to ensure they will work on all available hardware devices.*

Note: *If the hardware has not been detected when this call is made, the MGL will automatically detect the installed hardware the first time this function is called.*

See Also

MGL_init, MGL_availablePages, MGL_modeResolution, MGL_modeFlags, MGL_createDisplayDC, MGL_findMode

MGL_availableBitmap

Determines if the specified bitmap file is available for use.

Declaration

```
ibool MGLAPI MGL_availableBitmap(  
    const char *bitmapName)
```

Prototype In

mgraph.h

Parameters

bitmapName Name of bitmap file to check for

Return Value

True if the bitmap file exists, false if not.

Description

Attempt to locate the specified bitmap file, and verify that it is available for use. See *MGL_loadBitmap* for more information on the algorithm that MGL uses when searching for bitmap files on disk.

See Also

MGL_loadBitmap

MGL_availableCursor

Determines if the specified cursor file is available for use.

Declaration

```
ibool MGLAPI MGL_availableCursor(  
    const char *cursorName)
```

Prototype In

mgraph.h

Parameters

cursorName Name of cursor file to check for

Return Value

True if the cursor file exists, false if not.

Description

Attempt to locate the specified mouse cursor, and verify that it is available for use. See *MGL_loadCursor* for more information on the algorithm that MGL uses when searching for mouse cursor files on disk.

See Also

MGL_loadCursor

MGL_availableFont

Determines if a specific font file is available for use.

Declaration

```
ibool MGLAPI MGL_availableFont(  
    const char *fontname)
```

Prototype In

mgraph.h

Parameters

fontname Relative filename of the required font file

Return Value

True if font file is available, false if not.

Description

Attempt to locate the specified font file, and verify that it is available for use. See *MGL_loadFont* for more information on the algorithm that MGL uses when searching for font files on disk.

See Also

MGL_loadFont

MGL_availableIcon

Determines if the specified icon file is available for use.

Declaration

```
ibool MGLAPI MGL_availableIcon(  
    const char *iconName)
```

Prototype In

mgraph.h

Parameters

iconName Name of icon file to check for

Return Value

True if the icon file exists, False if not.

Description

Attempt to locate the specified icon file, and verify that it is available for use.

See Also

MGL_loadIcon

MGL_availableJPEG

Determines if the specified JPEG bitmap file is available for use.

Declaration

```
ibool MGLAPI MGL_availableJPEG(  
    const char *JPEGName)
```

Prototype In

mgraph.h

Parameters

JPEGName Name of JPEG bitmap file to check for

Return Value

True if the JPEG bitmap file exists, false if not.

Description

Attempt to locate the specified JPEG file, and verify that it is available for use. See *MGL_loadJPEG* for more information on the algorithm that MGL uses when searching for bitmap files on disk.

See Also

MGL_loadJPEG

MGL_availablePCX

Determines if the specified PCX bitmap file is available for use.

Declaration

```
ibool MGLAPI MGL_availablePCX(  
    const char *PCXName)
```

Prototype In

mgraph.h

Parameters

PCXName Name of PCX bitmap file to check for

Return Value

True if the PCX bitmap file exists, false if not.

MGL_availablePNG

Determines if the specified PNG bitmap file is available for use.

Declaration

```
ibool MGLAPI MGL_availablePNG(  
    const char *PNGName)
```

Prototype In

mgraph.h

Parameters

PNGName Name of PNG bitmap file to check for

Return Value

True if the a PNG bitmap file exists, false if not.

Description

Attempt to locate the specified PNG file, and verify that it is available for use. If the file exists the routine checks the signature to verify that it is really a PNG file. See *MGL_loadPNG* for more information on the algorithm that MGL uses when searching for bitmap files on disk.

See Also

MGL_loadPNG

MGL_availablePages

Determine the number of available video pages for a specific graphics mode.

Declaration

```
int MGLAPI MGL_availablePages(
    int mode)
```

Prototype In

mgraph.h

Parameters

mode MGL mode number to query

Return Value

Number of available display pages for mode, -1 for invalid mode number.

Description

Returns the number of pages of physical display pages available for a specific MGL graphics mode. You may call this routine before creating a display device context with *MGL_createDisplayDC*, but you cannot call this function until after you have called *MGL_init*. This function allows you to ignore display modes that do not have the required number of hardware display pages that your application requires.

Note: *If the hardware has not been detected when this call is made, the MGL will automatically detect the installed hardware the first time this function is called.*

See Also

MGL_init, *MGL_modeResolution*, *MGL_findMode*, *MGL_addCustomMode*,
MGL_createDisplayDC

MGL_backfacing

Determines if a polygon is backfacing.

Declaration

```
int MGLAPI MGL_backfacing(  
    fix32_t dx1,  
    fix32_t dy1,  
    fix32_t dx2,  
    fix32_t dy2)
```

Prototype In

mgraph.h

Parameters

<i>dx1</i>	change in x along first edge
<i>dy1</i>	change in y along first edge
<i>dx2</i>	change in x along second edge
<i>dy2</i>	change in y along second edge

Return Value

1 if the polygon is backfacing, 0 if it is frontfacing

Description

Determine whether a polygon is backfacing given two fixed point vectors. The vectors need to be derived from two consecutive counterclockwise edges of the polygon in order for this function to return accurate results.

Note that this function is written to correctly calculate the results for screen space coordinates, which can cause overflow with a normal 16.16 fixed point multiply if this is calculated directly using calls to *MGL_FixMul*.

MGL_beginDirectAccess

Enables direct framebuffer access.

Declaration

```
void MGLAPI MGL_beginDirectAccess(void)
```

Prototype In

mgraph.h

Description

Enables direct framebuffer access so that you can directly rasterize to the linear framebuffer memory using your own custom routines. Note that calling this function is absolutely necessary when using hardware acceleration, as this function correctly arbitrates between the hardware accelerator graphics engine and your direct framebuffer rasterizing code.

See Also

MGL_beginDirectAccess, *MGL_endDirectAccess*, *MGL_beginDirectAccessDC*,
MGL_endDirectAccessDC

MGL_beginDirectAccessDC

Enables direct framebuffer access (device context specific)

Declaration

```
void MGLAPI MGL_beginDirectAccessDC(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context to enable direct access to

Description

This is the same as *MGL_beginDirectAccess* but it takes a device context pointer instead of requiring the device context to be the current device context.

See Also

MGL_beginDirectAccess, *MGL_endDirectAccess*, *MGL_beginDirectAccessDC*,
MGL_endDirectAccessDC

MGL_beginPaint

Associate a window manager device context with an MGL device context.

Declaration

```
ibool MGLAPI MGL_beginPaint(
    MGLDC *dc,
    MGL_HDC hdc)
```

Prototype In

mglwin.h

Parameters

dc MGL windowed device context to use
hdc Handle to window manager device context to associate

Return Value

True if the application's palette has changed, false if not.

Description

This function and its and the corresponding function *MGL_endPaint()* should be called between the windows BeginPaint and EndPaint messages. This function allows MGL to use the newest clipping regions and viewport settings.

MGL_beginPaint() and *MGL_endPaint()* must bracket drawing functions that draw to a window type with a style of CS_PARENTDC or CS_CLASSDC. Such as dialog box controls. These types of windows allocate device handles on the fly so the HDC may change between calls to GetDC() or BeginPaint(). Therefore MGL cannot draw to these types of windows without knowing the new HDC after every BeginPaint() or GetDC() call.

OpenGL windows should NOT use *MGL_beginPaint* and *MGL_endPaint*.

A typical Windows WM_PAINT handler would be coded as follows:

```
case WM_PAINT:
    hdc = BeginPaint(hwnd, &ps);
    MGL_beginPaint(dc, hdc);
    // Do rasterizing code in here //
    MGL_bitBlt(dc, memDC, r, 0, 0, MGL_REPLACE_MODE);
    MGL_endPaint(dc);
    EndPaint(hwnd, &ps);
    return 0;
```

See Also

MGL_EndPaint

MGL_beginPixel

Setup for high speed pixel drawing.

Declaration

```
void MGLAPI MGL_beginPixel(void)
```

Prototype In

mgraph.h

Description

Sets up the video hardware for plotting single pixels as fast a possible. You must call this routine before calling any of the *MGL_pixel* and *MGL_getPixel* routines to ensure correct operation, and you must call the *MGL_endPixel* routine after you have finished.

This routine is intended primarily to ensure fast operation if you need to plot more than a single pixel at a time.

See Also

MGL_endPixel, *MGL_pixel*, *MGL_getPixel*.

MGL_bitBlt

Blts a block of image data from one device context into another.

Declaration

```
void MGL_bitBlt(
    MGLDC *dst,
    MGLDC *src,
    rect_t r,
    int dstLeft,
    int dstTop,
    int op)
```

Prototype In

mgraph.h

Parameters

<i>dst</i>	Destination device context
<i>src</i>	Source device context
<i>r</i>	Rectangle defining are to Blt from
<i>dstLeft</i>	Left coordinate to Blt to
<i>dstTop</i>	Right coordinate to Blt to
<i>op</i>	Write mode to use during Blt

Description

This function is the same as *MGL_bitBltCoord*, however it takes entire rectangles as parameters instead of coordinates.

See Also

MGL_bitBlt, *MGL_bitBltCoord*, *MGL_srcTransBlt*, *MGL_srcTransBltCoord*,
MGL_dstTransBlt, *MGL_dstTransBltCoord*, *MGL_bitBltPatt*, *MGL_bitBltPattCoord*,
MGL_bitBltFx, *MGL_bitBltFxCoord*, *MGL_stretchBlt*, *MGL_stretchBltCoord*,
MGL_stretchBltFx, *MGL_stretchBltFxCoord*, *MGL_copyPage*, *MGL_copyPageCoord*

MGL_bitBltCoord

Blts a block of image data from one device context into another.

Declaration

```
void MGLAPI MGL_bitBltCoord(
    MGLDC *dst,
    MGLDC *src,
    int left,
    int top,
    int right,
    int bottom,
    int dstLeft,
    int dstTop,
    int op)
```

Prototype In

mgraph.h

Parameters

<i>dst</i>	Destination device context
<i>src</i>	Source device context
<i>left</i>	Left coordinate of image to Blt from
<i>top</i>	Top coordinate of image to Blt from
<i>right</i>	Right coordinate of image to Blt from
<i>bottom</i>	Bottom coordinate of image to Blt from
<i>dstLeft</i>	Left coordinate to Blt to
<i>dstTop</i>	Right coordinate to Blt to
<i>op</i>	Write mode to use during Blt

Description

Copies a block of bitmap data from one device context to another. The source and destination rectangles may overlap even if the source and destination device contexts are the same, and MGL will correctly handle the overlapping regions. This routine has been highly optimized for absolute maximum performance, so it will provide the fastest method of copying bitmap data between device contexts. To obtain absolute maximum performance, you should align the source and destination bitmaps on DWORD boundaries (4 pixels for 8 bit, 2 pixels for 15/16 bit) and the low level device driver code will special case this for maximum performance.

This function will correctly handle Blt's across device contexts with differing pixel depths, and will perform the necessary pixel format translation to convert from the source device to the destination device. Note that although the code to implement this is highly optimized, this can be a time consuming operation so you should attempt to pre-convert all bitmaps to the current display device pixel format for maximum performance if using this routine for sprite animation.

When this function is called for 4 and 8 bit source bitmaps being copied to either 4 or 8bpp destination device contexts, MGL first checks if the color palettes for the source and destination bitmaps are the same. If they are not, MGL translates the pixel values from the source bitmap to the destination color palette, looking for the closest match color if an exact match is not found. In order to obtain maximum performance for blt'ing bitmaps in color index modes, you should ensure that the color palette in the source device matches the color palette in the destination device to avoid on the fly palette translation. If you know in advance that the palette is identical for a series of blit operations, you can turn off all identity palette checking in MGL with the *MGL_checkIdentityPalette* function.

When this function is called for 4 and 8 bit source bitmaps being copied to RGB destination device contexts, MGL will convert the pixels in the source bitmap using the source bitmap palette to map them to the destination pixel format. If however you know in advance that the palette for all source bitmaps is identical for a series of blit operations, you can use the *MGL_checkIdentityPalette* function to disable source palette translation. In this case the MGL will translate all color index bitmaps using the pre-defined color translation palette stored in the destination device context. You would then set the destination device context palette to the common palette for all blit operations using *MGL_setPalette*. If you are translating a lot of color index bitmaps, this will increase performance by avoiding the need to convert the palette entries to the destination pixel format for every blit operation.

This routine can also be used to perform hardware accelerated Blt's between offscreen memory devices and the display device when running in fullscreen modes, providing the hardware accelerator (if present) can support this operation.

The write mode operation specifies how the source image data should be combined with the destination image data. Write modes supported by the SciTech MGL are enumerated in *MGL_writeModeType*.

The source and destination rectangles are clipped according to the current clipping rectangles for the source and destination device contexts respectively.

See Also

MGL_bitBlit, *MGL_bitBlitCoord*, *MGL_srcTransBlit*, *MGL_srcTransBlitCoord*,
MGL_dstTransBlit, *MGL_dstTransBlitCoord*, *MGL_bitBlitPatt*, *MGL_bitBlitPattCoord*,
MGL_bitBlitFx, *MGL_bitBlitFxCoord*, *MGL_stretchBlit*, *MGL_stretchBlitCoord*,
MGL_stretchBlitFx, *MGL_stretchBlitFxCoord*, *MGL_copyPage*, *MGL_copyPageCoord*

MGL_bitBltFx

Copies a block of image data from one device context into another, while applying different effects in the process.

Declaration

```
void MGL_bitBltFx(
    MGLDC *dst,
    MGLDC *src,
    rect_t r,
    int dstLeft,
    int dstTop,
    bltfx_t *fx)
```

Prototype In

mgraph.h

Parameters

<i>dst</i>	Destination device context
<i>src</i>	Source device context
<i>r</i>	Rectangle defining are to Blt from
<i>dstLeft</i>	Left coordinate to Blt to
<i>dstTop</i>	Right coordinate to Blt to
<i>fx</i>	Information describing the effects to apply to the blit (<i>bltfx_t</i>)

Description

This function is the same as *MGL_bitBltFxCoord*, however it takes entire rectangles as parameters instead of coordinates.

See Also

MGL_bitBlt, *MGL_bitBltCoord*, *MGL_srcTransBlt*, *MGL_srcTransBltCoord*,
MGL_dstTransBlt, *MGL_dstTransBltCoord*, *MGL_bitBltPatt*, *MGL_bitBltPattCoord*,
MGL_bitBltFx, *MGL_bitBltFxCoord*, *MGL_stretchBlt*, *MGL_stretchBltCoord*,
MGL_stretchBltFx, *MGL_stretchBltFxCoord*, *MGL_copyPage*, *MGL_copyPageCoord*

MGL_bitBltFxCoord

Copies a block of image data from one device context into another, while applying different effects in the process.

Declaration

```
void MGLAPI MGL_bitBltFxCoord(
    MGLDC *dst,
    MGLDC *src,
    int left,
    int top,
    int right,
    int bottom,
    int dstLeft,
    int dstTop,
    bltfx_t *fx)
```

Prototype In

mgraph.h

Parameters

<i>dst</i>	Destination device context
<i>src</i>	Source device context
<i>left</i>	Left coordinate of image to Blt from
<i>top</i>	Top coordinate of image to Blt from
<i>right</i>	Right coordinate of image to Blt from
<i>bottom</i>	Bottom coordinate of image to Blt from
<i>dstLeft</i>	Left coordinate to Blt to
<i>dstTop</i>	Right coordinate to Blt to
<i>fx</i>	Information describing the effects to apply to the blit (<i>bltfx_t</i>)

Description

Copies a block of bitmap data from one device context to another, with optional effects applied. The effects applied to the blit operation range from X and Y bitmap flipping to color transparency and blending. All of these operations can be applied to data being copied between different device context of different color depths and pixel formats if desired (ie: 32-bit ARGB alpha blended bitmap to a 16-bit device context). Please refer to the documentation for the *bltfx_t* structure and the *MGL_bitBltFxFlagsType* enumeration that defines the flags passed to this function.

This function will correctly handle effects Blt's across device contexts with differing pixel depths, and will perform the necessary pixel format translation to convert from the source device to the destination device. Note that although the code to implement this is highly optimized, this can be a time consuming operation so you should attempt to pre-convert all bitmaps to the current display device pixel format for maximum performance if using this routine for sprite animation.

When this function is called for 4 and 8 bit source bitmaps being copied to either 4 or 8bpp destination device contexts, MGL first checks if the color palettes for the source and destination bitmaps are the same. If they are not, MGL translates the pixel values from the source bitmap to the destination color palette, looking for the closest match color if an exact match is not found. In order to obtain maximum performance for blt'ing bitmaps in color index modes, you should ensure that the color palette in the source device matches the color palette in the destination device to avoid on the fly palette translation. If you know in advance that the palette is identical for a series of blit operations, you can turn off all identity palette checking in MGL with the *MGL_checkIdentityPalette* function.

When this function is called for 4 and 8 bit source bitmaps being copied to RGB destination device contexts, MGL will convert the pixels in the source bitmap using the source bitmap palette to map them to the destination pixel format. If however you know in advance that the palette for all source bitmaps is identical for a series of blit operations, you can use the *MGL_checkIdentityPalette* function to disable source palette translation. In this case the MGL will translate all color index bitmaps using the pre-defined color translation palette stored in the destination device context. You would then set the destination device context palette to the common palette for all blit operations using *MGL_setPalette*. If you are translating a lot of color index bitmaps, this will increase performance by avoiding the need to convert the palette entries to the destination pixel format for every blit operation.

The source and destination rectangles are clipped according to the current clipping rectangles for the source and destination device contexts respectively.

Note: *This function is not designed to support overlapping source and destination rectangles on the same device context so if the source and destination rectangles overlap on the same device context, the results are undefined.*

See Also

MGL_bitBlt, MGL_bitBltCoord, MGL_srcTransBlt, MGL_srcTransBltCoord, MGL_dstTransBlt, MGL_dstTransBltCoord, MGL_bitBltPatt, MGL_bitBltPattCoord, MGL_bitBltFx, MGL_bitBltFxCoord, MGL_stretchBlt, MGL_stretchBltCoord, MGL_stretchBltFx, MGL_stretchBltFxCoord, MGL_copyPage, MGL_copyPageCoord

MGL_bitBltPatt

Blts a block of image data from one device context into another while applying a mono or color pattern.

Declaration

```
void MGL_bitBltPatt(
    MGLDC *dst,
    MGLDC *src,
    rect_t r,
    int dstLeft,
    int dstTop,
    int usePixmap,
    int rop3)
```

Prototype In

mgraph.h

Parameters

<i>dst</i>	Destination device context
<i>src</i>	Source device context
<i>r</i>	Rectangle defining are to Blt from
<i>dstLeft</i>	Left coordinate to Blt to
<i>dstTop</i>	Right coordinate to Blt to
<i>usePixmap</i>	True to use color pixmap pattern, false for mono bitmap pattern
<i>rop3</i>	ROP3 raster operation code to use during Blt (<i>MGL_rop3CodesType</i>)

Description

This function is the same as *MGL_bitBltPattCoord*, however it takes entire rectangles as parameters instead of coordinates.

See Also

MGL_bitBlt, *MGL_bitBltCoord*, *MGL_srcTransBlt*, *MGL_srcTransBltCoord*, *MGL_dstTransBlt*, *MGL_dstTransBltCoord*, *MGL_bitBltPatt*, *MGL_bitBltPattCoord*, *MGL_bitBltFx*, *MGL_bitBltFxCoord*, *MGL_stretchBlt*, *MGL_stretchBltCoord*, *MGL_stretchBltFx*, *MGL_stretchBltFxCoord*, *MGL_copyPage*, *MGL_copyPageCoord*

MGL_bitBltPattCoord

Blts a block of image data from one device context into another while applying a mono or color pattern.

Declaration

```
void MGLAPI MGL_bitBltPattCoord(
    MGLDC *dst,
    MGLDC *src,
    int left,
    int top,
    int right,
    int bottom,
    int dstLeft,
    int dstTop,
    int usePixmap,
    int rop3)
```

Prototype In

mgraph.h

Parameters

<i>dst</i>	Destination device context
<i>src</i>	Source device context
<i>left</i>	Left coordinate of image to Blt from
<i>top</i>	Top coordinate of image to Blt from
<i>right</i>	Right coordinate of image to Blt from
<i>bottom</i>	Bottom coordinate of image to Blt from
<i>dstLeft</i>	Left coordinate to Blt to
<i>dstTop</i>	Right coordinate to Blt to
<i>usePixmap</i>	True to use color pixmap pattern, false for mono bitmap pattern
<i>rop3</i>	ROP3 raster operation code to use during Blt (<i>MGL_rop3CodesType</i>)

Description

Copies a block of bitmap data from one device context to another, while applying either a mono bitmap pattern or a color pixmap pattern to the data with a ternary raster operation code (ROP3). If the usePixmap parameter is set to true, the current pixmap pattern set by *MGL_setPenPixmapPattern* will be applied as pattern data, otherwise the current monochrome bitmap pattern set by *MGL_setPenBitmapPattern* will be applied.

The source and destination rectangles may overlap even if the source and destination device contexts are the same, and MGL will correctly handle the overlapping regions.

This function will only work with Blt's between device contexts that have identical pixel formats. If the color depth or pixel formats are different, this function will produce undefined results.

The ROP3 code specifies how the source, pattern and destination image data should be combined to produce the final result. SciTech MGL supports all 256 ROP3 codes, and they are enumerated in *MGL_rop3CodesType*.

The source and destination rectangles are clipped according to the current clipping rectangles for the source and destination device contexts respectively.

Note: *This function is not designed to support overlapping source and destination rectangles on the same device context so if the source and destination rectangles overlap on the same device context, the results are undefined.*

See Also

MGL_bitBlt, MGL_bitBltCoord, MGL_srcTransBlit, MGL_srcTransBlitCoord, MGL_dstTransBlit, MGL_dstTransBlitCoord, MGL_bitBltPatt, MGL_bitBltPattCoord, MGL_bitBltFx, MGL_bitBltFxCoord, MGL_stretchBlit, MGL_stretchBlitCoord, MGL_stretchBltFx, MGL_stretchBltFxCoord, MGL_copyPage, MGL_copyPageCoord

MGL_buildMonoMask

Create a monochrome bitmap mask given a transparent color.

Declaration

```
bitmap_t * MGLAPI MGL_buildMonoMask(
    bitmap_t *bitmap,
    color_t transparent)
```

Prototype In

mgraph.h

Parameters

<i>bitmap</i>	bitmap to build monochrome mask from
<i>transparent</i>	transparent color to mask out

Return Value

Pointer to allocated bitmap mask, or NULL on error.

Description

Attempts to build a monochrome bitmap mask that can be used to rasterize transparent bitmaps. This is useful for devices that have hardware BitBlt capabilities but lack hardware transparent BitBlt support. Everywhere the transparent color is found, the mask will be 0, and everywhere else it will be 1. Once you have created a monochrome bitmap mask, you can rasterize a transparent bitmap by replacing all the transparent pixels in the original bitmap with 0's, and then doing the following:

1. Set the foreground color to black and drawing the bitmap mask with *MGL_putBitmapMask*. Everywhere that the bitmap has data a black pixel will be used to 'punch' a hole in the display.
2. Use the *MGL_bitBlt* function to draw the original bitmap on the display using the *MGL_OR_MODE* write mode. Everywhere that a pixel is transparent in the original source bitmap, the destination will remain the same. Everywhere that a pixel is non-transparent the pixel on the destination will be replaced with the pixel in the source bitmap.

Note that only 8+ bits per pixel bitmaps are supported, and this function will fail on lower color depths.

See Also

MGL_transBltLin, *MGL_putBitmapMask*, *MGL_bitBlt*

MGL_charWidth

Returns the width of a character in pixels.

Declaration

```
int MGLAPI MGL_charWidth(  
    char ch)
```

Prototype In

mgraph.h

Parameters

ch Character to measure

Return Value

Width of the character in pixels (will depend on currently active font)

Description

Return the width of the specified character, given the currently active font and attribute settings.

See Also

MGL_textWidth, *MGL_textHeight*, *MGL_useFont*, *MGL_charWidth_W*

MGL_charWidth_W

Returns the width of a wide character in pixels.

Declaration

```
int MGLAPI MGL_charWidth_W(  
    wchar_t ch)
```

Prototype In

mgraph.h

Parameters

ch Wide character to measure

Return Value

Width of the wide character in pixels (will depend on currently active font)

Description

Return the width of the specified wide character, given the currently active font and attribute settings. This function is the same as *MGL_charWidth*, but provides support for Unicode wide characters (for far-east languages).

See Also

MGL_textWidth_W, *MGL_textHeight*, *MGL_useFont*, *MGL_charWidth*

MGL_checkIdentityPalette

Turns on or off identity palette checking.

Declaration

```
ibool MGLAPI MGL_checkIdentityPalette(  
    ibool enable)
```

Prototype In

mgraph.h

Parameters

enable True to enable identity palette checking, false to disable

Return Value

Old value of the identity palette check flag.

Description

Turns on or off the checking of identity palette mappings for MGL. This is a global flag, and by default, identity palette checking is turned on.

When any MGL blitting function is called for 4 and 8 bit source bitmaps being copied to either 4 or 8bpp destination device contexts, MGL first checks if the color palettes for the source and destination bitmaps are the same. If they are not, MGL translates the pixel values from the source bitmap to the destination color palette, looking for the closest match color if an exact match is not found. In order to obtain maximum performance for blt'ing bitmaps in color index modes, you should ensure that the color palette in the source device matches the color palette in the destination device to avoid on the fly palette translation. If you know in advance that the palette is identical for a series of blit operations, you can turn off all identity palette checking in MGL with the *MGL_checkIdentityPalette* function.

When any MGL blitting function called for 4 and 8 bit source bitmaps being copied to RGB destination device contexts, MGL will convert the pixels in the source bitmap using the source bitmap palette to map them to the destination pixel format. If however you know in advance that the palette for all source bitmaps is identical for a series of blit operations, you can use the *MGL_checkIdentityPalette* function to disable source palette translation. In this case the MGL will translate all color index bitmaps using the pre-defined color translation palette stored in the destination device context. You would then set the destination device context palette to the common palette for all blit operations using *MGL_setPalette*. If you are translating a lot of color index bitmaps, this will increase performance by avoiding the need to convert the palette entries to the destination pixel format for every blit operation.

See Also

MGL_bitBlt, *MGL_stretchBlt*, *MGL_putBitmap*

MGL_clearDevice

Clears the currently active display page.

Declaration

```
void MGLAPI MGL_clearDevice(void)
```

Prototype In

mgraph.h

Description

This function will clear the entire currently active display page in the current background color. This is the fastest way to clear an entire display page, but if you wish to only clear a portion of the page, use the *MGL_clearViewport* routine instead.

See Also

MGL_clearViewport

MGL_clearRegion

Clears the specified region to an empty region.

Declaration

```
void MGLAPI MGL_clearRegion(  
    region_t *r)
```

Prototype In

mgraph.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

MGL_newRegion, MGL_copyRegion, MGL_freeRegion

MGL_clearViewport

Clears the currently active viewport.

Declaration

```
void MGLAPI MGL_clearViewport(void)
```

Prototype In

mgraph.h

Description

This function will clear the currently active display page viewport to the current background color. This is the fastest way to clear a rectangular viewport, but you may also wish to use the *MGL_fillRect* routine to fill with an arbitrary pattern instead, as this function always clears the viewport to the solid background color.

See Also

MGL_clearDevice, *MGL_fillRect*

MGL_closeFontLib

Unloads a font file from memory.

Declaration

```
void MGLAPI MGL_closeFontLib(  
    font_lib_t *lib)
```

Prototype In

mgraph.h

Description

Unloads the specified font file from memory, and frees up all the system resources associated with this font.

See Also

MGL_openFontLib, MGL_openFontLibExt

MGL_computePixelAddr

Computes the address of a pixel in the device context surface.

Declaration

```
void * MGLAPI MGL_computePixelAddr(
    MGLDC *dc,
    int x,
    int y)
```

Prototype In mgraph.h

Parameters

<i>dc</i>	Device context to compute the pixel address for
<i>x</i>	X coordinate of pixel to address
<i>y</i>	Y coordinate of pixel to address

Return Value

Pointer to the start of the pixel in the surface of the device context.

Description

This function computes the address of a pixel in the surface of a specific device context. This function is most useful for developing custom rendering routines that draw directly to the surface of a device context, and will compute the address of the pixel correctly regardless of the color depth of the device context. Essentially this function computes the following:

```
addr = dc->surface + (y * bytesPerLine) + (x * bytesPerPixel)
```

If you are going to be doing a lot of address calculations, it will be faster to optimise your code to do the calculations directly in place (such as with a macro) and specifically to eliminate the last multiply if you know in advance what color depth you are working with (ie: change it to be x, x*2, x*3 or x*4 depending on the color depth).

Note: *You cannot use this function to address the device context surface if the device surface access type returned by MGL_getSurfaceAccessType is set to MGL_NO_ACCESS.*

MGL_copyBitmapToBuffer

Copies a section of a bitmap to an offscreen buffer

Declaration

```
void MGLAPI MGL_copyBitmapToBuffer(
    bitmap_t *bitmap,
    int left,
    int top,
    int right,
    int bottom,
    int dstLeft,
    int dstTop,
    MGLBUF *buf)
```

Prototype In

mgraph.h

Parameters

<i>bitmap</i>	Bitmap to blit from
<i>left</i>	Left coordinate in device context to blit from
<i>top</i>	Top coordinate in device context to blit from
<i>right</i>	Right coordinate in device context to blit from
<i>bottom</i>	Bottom coordinate in device context to blit from
<i>dstLeft</i>	Left coordinate in buffer to blit to
<i>dstTop</i>	Top coordinate in buffer to blit to
<i>buf</i>	MGL buffer to blit to

Description

This function is used to copy bitmap data from a lightweight system memory bitmap into an offscreen buffer. The information copied is clipped to the full dimensions of the buffer.

See Also

MGL_copyToBuffer, *MGL_putBuffer*

MGL_copyIntoRegion

Copy the contents of one region into another region.

Declaration

```
void MGLAPI MGL_copyIntoRegion(  
    region_t *d,  
    const region_t *s)
```

Prototype In

mgraph.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 *MGL_copyRegion*, however it does not allocate a new region but rather copies the data into an existing region.

See Also

MGL_newRegion, *MGL_freeRegion*, *MGL_clearRegion*, *MGL_copyRegion*

MGL_copyPage

Blts a block of image data from one page in a display device context to another page within the same device context.

Declaration

```
void MGL_copyPage(
    MGLDC *dc,
    int srcPage,
    rect_t r,
    int dstLeft,
    int dstTop,
    int op)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Destination device context
<i>srcPage</i>	Page in DC to get the source data from
<i>r</i>	Rectangle defining are to Blt from
<i>dstLeft</i>	Left coordinate to Blt to
<i>dstTop</i>	Right coordinate to Blt to
<i>op</i>	Write mode to use during Blt

Description

This function is the same as *MGL_copyPageCoord*, however it takes entire rectangles as parameters instead of coordinates.

See Also

MGL_bitBlt, *MGL_bitBltCoord*, *MGL_srcTransBlt*, *MGL_srcTransBltCoord*,
MGL_dstTransBlt, *MGL_dstTransBltCoord*, *MGL_bitBltPatt*, *MGL_bitBltPattCoord*,
MGL_bitBltFx, *MGL_bitBltFxCoord*, *MGL_stretchBlt*, *MGL_stretchBltCoord*,
MGL_stretchBltFx, *MGL_stretchBltFxCoord*, *MGL_copyPage*, *MGL_copyPageCoord*

MGL_copyPageCoord

Blts a block of image data from one page in a display device context to another page within the same device context.

Declaration

```
void MGLAPI MGL_copyPageCoord(
    MGLDC *dc,
    int srcPage,
    int left,
    int top,
    int right,
    int bottom,
    int dstLeft,
    int dstTop,
    int op)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Destination device context
<i>srcPage</i>	Page in DC to get the source data from
<i>left</i>	Left coordinate of image to Blt from
<i>top</i>	Top coordinate of image to Blt from
<i>right</i>	Right coordinate of image to Blt from
<i>bottom</i>	Bottom coordinate of image to Blt from
<i>dstLeft</i>	Left coordinate to Blt to
<i>dstTop</i>	Right coordinate to Blt to
<i>op</i>	Write mode to use during Blt

Description

Copies a block of bitmap data from the source page in the destination device context to the currently active page. This routine has been highly optimized for absolute maximum performance, so it will provide the fastest method of copying bitmap data between two different pages in a display device context.

The write mode operation specifies how the source image data should be combined with the destination image data. Write modes supported by the SciTech MGL are enumerated in *MGL_writeModeType*.

The destination rectangle is clipped according to the current clipping rectangles for the destination device context. However the source rectangle is only clipped to the bounds of the source page dimensions and is not clipped to the source clip rectangle. Also the source coordinates are global screen coordinates and are not viewport translated. You will need to do your own viewport translation as necessary.

Note: *This function will utilise the hardware whenever available to speed up the blitting of data between display pages. However if the end user system does not have hardware screen to screen blit operations (or the underlying device driver does not support this), then the copy operation will be quite slow. It would be faster in this case to do all rendering to a system memory DC and blit that to the screen instead of doing software blit's between two display pages.*

See Also

MGL_bitBlt, MGL_bitBltCoord, MGL_srcTransBlt, MGL_srcTransBltCoord,
MGL_dstTransBlt, MGL_dstTransBltCoord, MGL_bitBltPatt, MGL_bitBltPattCoord,
MGL_bitBltFx, MGL_bitBltFxCoord, MGL_stretchBlt, MGL_stretchBltCoord,
MGL_stretchBltFx, MGL_stretchBltFxCoord, MGL_copyPage, MGL_copyPageCoord

MGL_copyRegion

Create a copy of the specified region.

Declaration

```
region_t * MGLAPI MGL_copyRegion(  
    const region_t *s)
```

Prototype In

mgraph.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 MGL 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

MGL_newRegion, MGL_freeRegion, MGL_clearRegion, MGL_copyIntoRegion

MGL_copyToBuffer

Copies a section of a device context to an offscreen buffer

Declaration

```
void MGLAPI MGL_copyToBuffer (
    MGLDC *dc,
    int left,
    int top,
    int right,
    int bottom,
    int dstLeft,
    int dstTop,
    MGLBUF *buf)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to blit from
<i>left</i>	Left coordinate in device context to blit from
<i>top</i>	Top coordinate in device context to blit from
<i>right</i>	Right coordinate in device context to blit from
<i>bottom</i>	Bottom coordinate in device context to blit from
<i>dstLeft</i>	Left coordinate in buffer to blit to
<i>dstTop</i>	Top coordinate in buffer to blit to
<i>buf</i>	MGL buffer to blit to

Description

This function is used to copy bitmap data from an MGL device context into an offscreen buffer. The information copied is clipped to the full dimensions of the buffer, and the source clipping rectangle in the source device context is ignored.

See Also

MGL_copyBitmapToBuffer, *MGL_putBuffer*

MGL_createBuffer

Creates a new MGL memory buffer for storing bitmaps

Declaration

```
MGLBUF * MGLAPI MGL_createBuffer(
    MGLDC *dc,
    int width,
    int height,
    M_uint32 flags)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	MGL display or windowed device context to allocate buffer from
<i>width</i>	Width of the buffer in pixels
<i>height</i>	Height of the buffer in scanlines
<i>flags</i>	Flags to use when creating the buffer

Return Value

Pointer to the allocated buffer, NULL on failure.

Description

This function allocates a new lightweight bitmap buffer in offscreen display memory. Lightweight bitmap buffers are used to store bitmap and sprite information in the offscreen display memory in the hardware, but are not full device contexts. Hence buffers have much less memory overhead than a full offscreen device context. Buffers can only be used for storing bitmaps and blitting them around on the screen. You can copy the contents to a MGL device context using the *MGL_putBuffer*, *MGL_stretchBuffer* and *MGL_putBufferSrcTrans* functions. You can also copy the contents of an MGL device context to a buffer using the *MGL_copyToBuffer* function.

If you need to draw on a buffer in offscreen memory, create a full offscreen device context instead. Then you can call any of the standard MGL drawing functions and BitBlt operations for the offscreen memory buffer. The primary disadvantage of doing this is that a full offscreen device context has a lot more memory overhead involved in maintaining the device context state information than a simple offscreen buffer.

If you flag the MGL_BUF_CACHED flag, MGL will always keep a cached copy of the surface memory for the buffer in system memory. If your offscreen buffers get lost (ie: on a fullscreen mode switch), they can be automatically be restored when the application regains the active focus. This also allows the MGL to compact the offscreen memory heap when necessary. If you don't set the *cached* parameter to true, then it is your applications responsibility to reload the bitmaps when the focus is lost and regained. Note however that the MGL is *not* responsible for maintaining the contents of the buffer cache memory, so if you draw directly on the buffer surface you should

ensure that the cache is kept up to date if you wish for your buffers to be cached. Note that the *MGL_copyToBuffer* and *MGL_copyBitmapToBuffer* functions will ensure that both the buffer cache and video memory buffer contents are updated if the buffer is cached. Hence if you are only using the buffers to store static sprite information, you can use *MGL_copyToBuffer()* or *MGL_copyBitmapToBuffer()* and never have to worry about keeping the buffer cache in sync.

Note: *The MGL automatically manages offscreen display memory, and if you run out of offscreen display memory it will place the buffer surfaces in system memory (unless you pass the MGL_BUF_NOSYSMEM flag). Hence you should allocate your important buffers first, to ensure they end up in offscreen memory for speedy drawing.*

Note: *This function is only valid for display device contexts, and will fail if you call it for a different device context type with an error code of grInvalidDC.*

See Also

MGL_createBuffer, MGL_lockBuffer, MGL_unlockBuffer, MGL_destroyBuffer, MGL_createOffscreenDC, MGL_putBuffer, MGL_stretchBuffer, MGL_putBufferSrcTrans, MGL_putBufferDstTrans, MGL_copyToBuffer, MGL_copyBitmapToBuffer, MGL_updateBufferCache, MGL_updateFromBufferCache

MGL_createCustomDC

Creates a new custom memory device context.

Declaration

```
MGLDC * MGLAPI MGL_createCustomDC (
    int xSize,
    int ySize,
    int bitsPerPixel,
    pixel_format_t *pf,
    int bytesPerLine,
    void *surface,
    MGL_HBITMAP hbm)
```

Prototype In

mgraph.h

Parameters

<i>xSize</i>	X resolution for the memory context
<i>ySize</i>	Y resolution for the memory context
<i>bitsPerPixel</i>	Pixel depth for the memory context
<i>pf</i>	Pixel format for memory context
<i>bytesPerLine</i>	Buffer pitch for memory context
<i>surface</i>	Pointer to surface memory for context
<i>hbm</i>	Handle to HBITMAP for DIB section

Return Value

Pointer to the allocated memory device context.

Description

This function is useful for creating an MGL device context for a memory block provided by an application. This allows the SciTech MGL to render to memory it does not own (e.g. custom hardware framebuffers, or bitmaps allocated by other runtime libraries).

For Windows if the hbm parameter is not NULL, it is assumed that the original memory was created by a call to CreateDIBSection and the hbm parameter is a handle to the bitmap object for the DIB section. This parameter can be used to blit the memory DC image to a windowed DC using the standard Windows GDI blit functions. The hbm parameter is not necessary for fullscreen modes.

See Also

MGL_createMemoryDC, *MGL_destroyDC*

MGL_createDisplayDC

Create a new display device context.

Declaration

```
MGLDC * MGLAPI MGL_createDisplayDC (
    int mode,
    int numBuffers,
    int refreshRate)
```

Prototype In

mgraph.h

Parameters

<i>mode</i>	Graphics mode to initialise
<i>numBuffers</i>	Number of buffers to allocate for double/multi-buffering.
<i>refreshRate</i>	Requested refresh rate for the graphics mode

Return Value

Pointer to the newly created display device context, NULL on failure

Description

Creates a new display device context for drawing information directly to the hardware display device in fullscreen graphics modes. When the device context is created, the MGL will start the graphics mode specified in the mode parameter and initialize the specific device driver. If any prior display device contexts exist, they will all be destroyed before switching to the new display mode.

If you intend to use double or multi-buffered graphics using the display device, you should set the numBuffers to the number of buffers that you require, so that the device will be properly configured for multi-buffered operation. If you request more buffers than is currently available, this function will fail. Hence you should first call *MGL_availablePages* to determine how many buffers can be used in the desired graphics mode.

The refresh rate value that you pass in is a *suggested* value in that the MGL will attempt to set the refresh rate to this value, however if the hardware does not support that refresh rate the next lowest available refresh rate will be used instead. In some situations where no refresh rate control is available, the value will be ignored and the adapter default refresh rate will be used. If you don't care about the refresh rate and want to use the adapter default setting, pass in a value of MGL_DEFAULT_REFRESH.

Once the display device context has been allocated, the surface pointer of the *MGLDC* structure can be used to directly access the surface of the device context as a linear block of memory. The dimensions and pixel format of the device context surface are stored in the *gmode_t* field of the *MGLDC* structure, and you should use these values to write your own direct rendering code.

Note that all device contexts have an associated color palette, even RGB device contexts. In RGB modes the color palette is used for converting color index pixel values to RGB values during BitBlt operations and with the *MGL_realColor* and *MGL_setColorCI* function.

Note: *To set an interlaced refresh rate, pass in the refresh rate as a negative value. Ie: Pass a value of -87 for 87Hz interlaced.*

See Also

MGL_createMemoryDC, *MGL_createScrollingDisplayDC*, *MGL_createStereoDisplayDC*, *MGL_destroyDC*

MGL_createMemoryDC

Create a new memory device context.

Declaration

```
MGLDC * MGLAPI MGL_createMemoryDC (
    int xSize,
    int ySize,
    int bitsPerPixel,
    pixel_format_t *pf)
```

Prototype In

mgraph.h

Parameters

<i>xSize</i>	x resolution for the memory context
<i>ySize</i>	y resolution for the memory context
<i>bitsPerPixel</i>	Pixel depth for the memory context
<i>pf</i>	Pixel format for memory context (NULL for 8 bits and below)

Return Value

Pointer to the allocated memory device context, NULL if not enough memory.

Description

Creates a new memory device context, allocating the necessary memory resources to hold the surface of the memory device given the specified resolution and pixel depth. The surface of a memory device context is always allocated using the appropriate operating system specific functions, and you can always directly access the surface of the device context via the surface pointer of the *MGLDC* structure. If you do directly access the surface, the dimensions and pixel format of the device context surface are stored in the *gmode_t* field of the *MGLDC* structure, and you should use these values to write your own direct rendering code.

For memory device contexts with pixel depths greater than 8 bits per pixel, you must also pass a valid *pixel_format_t* structure which defines the pixel format to be used for the device context. If you wish to create a memory device context for your main rasterizing context which you then wish to Blt to the screen, you must ensure that you use the same pixel format for the memory device as the display device for the current graphics mode, otherwise the pixel formats will be translated on the fly by the *MGL_bitBlt* function resulting in very low performance. You can use the *MGL_getPixelFormat* function to obtain the pixel format information for the display device context you are using.

Note that all device contexts have an associated color palette, even RGB device contexts. In RGB modes the color palette is used for converting color index pixel values to RGB values during BitBlt operations and with the *MGL_realColor* and *MGL_setColorCI* function.

See Also

MGL_createCustomDC, *MGL_createDisplayDC*, *MGL_destroyDC*

MGL_createOffscreenDC

Creates a new offscreen display device context.

Declaration

```
MGLDC * MGLAPI MGL_createOffscreenDC (
    MGLDC *dc,
    int width,
    int height)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	MGL display or windowed device context to create offscreen DC from
<i>width</i>	Width of the offscreen device context to create
<i>height</i>	Height of the offscreen device context to create

Return Value

Pointer to the newly created offscreen device context, NULL if not valid.

Description

Creates a new offscreen display device context for rendering to offscreen video memory when running in hardware accelerated video modes. You must have already created a valid display device context before this function is called, otherwise this function will return NULL. Also if the display device does not support hardware accelerated offscreen display memory, this function will also return NULL. Finally if there is no more available offscreen display memory to fulfill the request, this function will fail. See the *MGL_result* error code for more information if the function failed.

An offscreen device context can be used just like any other MGL device context, so you can copy data around with *MGL_bitBlt*, or draw on the device context using any of the MGL rendering functions. If you simply want to store bitmaps in offscreen memory for fast blitting, you should instead create the bitmaps as MGL buffers with the *MGL_createBuffer* function. MGL buffers are similar to offscreen device contexts, except they can only be used for blitting and cannot be used as a rendering target. MGL buffers also use a lot less system memory resources than a full offscreen DC.

See Also

MGL_createMemoryDC, *MGL_createScrollingDisplayDC*, *MGL_createStereoDisplayDC*, *MGL_createDisplayDC*, *MGL_destroyDC*, *MGL_createBuffer*

MGL_createScrollingDC

Create a new hardware scrolling display device context.

Declaration

```
MGLDC * MGLAPI MGL_createScrollingDC (
    int mode,
    int virtualX,
    int virtualY,
    int numBuffers,
    int refreshRate)
```

Prototype In

mgraph.h

Parameters

<i>mode</i>	Graphics mode to initialise
<i>virtualX</i>	Virtual width of desired mode
<i>virtualY</i>	Virtual height of desired mode
<i>numBuffers</i>	Number of buffers for multibuffering
<i>refreshRate</i>	Requested refresh rate for the graphics mode

Return Value

Pointer to the newly created hardware scrolling display device context, NULL on failure.

Description

Creates a new scrolling display device context for drawing information directly to the hardware display device in fullscreen graphics modes. Essentially this function is identical to *MGL_createDisplayDC*, however hardware scrolling (or panning) is supported. Some hardware devices may not support hardware scrolling, in which case this function will fail and return a NULL. In these cases you should provide an alternative method of scrolling the display, such as drawing to a memory device context and copying the appropriate portion of the image to the display with *MGL_bitBlt*.

When the device context is created, the MGL will start the graphics mode specified in the mode parameter and initialize the specific device driver. If any prior display device contexts exist, they will all be destroyed before switching to the new display mode.

Once you have created a hardware scrolling device context, the display starting coordinate will be set to (0,0) within the virtual image. To hardware pan around within the virtual image, you can use the *MGL_setDisplayStart* function to change the display starting x and y coordinates.

The refresh rate value that you pass in is a *suggested* value in that the MGL will attempt to set the refresh rate to this value, however if the hardware does not support that refresh rate the next lowest available refresh rate will be used instead. In some situations where no refresh rate control is available, the value will be ignored and the adapter

default refresh rate will be used. If you dont care about the refresh rate and want to use the adapter default setting, pass in a value of MGL_DEFAULT_REFRESH.

Note: *To set an interlaced refresh rate, pass in the refresh rate as a negative value. Ie: Pass a value of -87 for 87Hz interlaced.*

See Also

*MGL_createMemoryDC, MGL_createDisplayDC, MGL_createStereoDisplayDC,
MGL_destroyDC*

MGL_createStereoDisplayDC

Create a new display device context for stereo LC shutter glasses

Declaration

```
MGLDC * MGLAPI MGL_createStereoDisplayDC(
    int mode,
    int numBuffers,
    int refreshRate)
```

Prototype In mgraph.h

Parameters

<i>mode</i>	Graphics mode to initialise
<i>numBuffers</i>	Number of buffers to allocate for double/multi-buffering.
<i>refreshRate</i>	Requested refresh rate for the graphics mode

Return Value

Pointer to the newly created display device context, NULL on failure.

Description

Creates a new display device context for drawing information directly to the hardware display device in fullscreen graphics modes. Essentially this function is identical to *MGL_createDisplayDC*, however support for LC shutter glasses is provided and the MGL will take care of automatically flipping between the left and right images to create the stereo display. In some cases we may not be able to initialise support for LC shutter glasses, so this function will fail.

When the device context is created, the MGL will start the graphics mode specified in the mode parameter and initialize the specific device driver. If any prior display device contexts exist, they will all be destroyed before switching to the new display mode.

When running in stereo mode, the MGL actually allocates twice the number of buffers that you request for drawing images, since we need one buffer for the left eye image and another buffer for the right eye image (ie: if you request two stereo buffers for double buffering, the MGL will actually allocate room for four). The reason for this is that when displaying one of the stereo buffers, the MGL will automatically *swap* between the left and right eye images at every vertical retrace. It also sends a signal to the LC shutter glasses to tell them to block out the image for the eye that should not be seeing the image on the screen (ie: when the left image is being displayed, the shutter over the right eye will be blacked out). Hence by drawing images with slightly different viewing parameters (ie: as viewed from the left or right eye when doing 3D rendering), the user sees a single image with complete with visual depth cues!

When running in stereo mode, you have to tell the MGL which buffer you want to draw to when drawing the left or right eye images. Just like you normally do in double and multi-buffering, you use the *MGL_setActivePage* function to tell the MGL the active

display page you wish to draw to. However in stereo modes you must also pass in the MGL_LEFT_BUFFER or MGL_RIGHT_BUFFER values to tell the MGL which eye you are drawing for. For instance to draw to stereo page 1, left eye you would use `MGL_setActivePage(1 | MGL_LEFT_BUFFER)`, and for the right eye you would use `MGL_setActivePage(1 | MGL_RIGHT_BUFFER)`.

Note: *In OpenGL rendering modes, changing the draw buffer is done with the OpenGL `glDrawBuffer(GL_BACK_LEFT)` and `glDrawBuffer(GL_BACK_RIGHT)` functions instead of using `MGL_setActivePage`.*

One of the biggest drawbacks to viewing stereo images using LC shutter glasses is that the refresh rate viewed in each eye is exactly half that of the refresh rate of the display mode. Hence if running in a display mode with a 60Hz refresh rate, the user will experience an overall refresh rate of 30Hz per eye! As you can imagine this can be extremely tiresome for extended viewing periods, so to get around this the MGL allows you to pass in a value to request a higher refresh rate for the mode. Ideally you want to try and use a refresh rate that is twice the desired refresh rate per eye, such as 120Hz for viewing images at 60Hz, however you *must* allow the user to override or suggest a desired refresh rate as many older monitors may not be capable of displaying an image at a high refresh rate like 120Hz.

The refresh rate value that you pass in is a *suggested* value in that the MGL will attempt to set the refresh rate to this value, however if the hardware does not support that refresh rate the next lowest available refresh rate will be used instead. In some situations where no refresh rate control is available, the value will be ignored and the adapter default refresh rate will be used. If you don't care about the refresh rate and want to use the adapter default setting, pass in a value of MGL_DEFAULT_REFRESH.

Note: *In the USA and Canada, the main power frequency runs at 60Hz, and all fluorescent lights will be illuminating your room at frequency of 60Hz. If you use a refresh rate that is not a multiple of the mains frequency and you are viewing the image in a room with fluorescent lights, you may experience severe beating at a frequency that is the difference between the monitor refresh rate and the fluorescent light frequency (ie: at 100Hz you will experience a 20Hz annoying beat frequency). In order to get around this problem, always try to use a frequency that is double the mains frequency such as 120Hz to avoid the beating, or have the user turn off their fluorescent lights!*

When you create a stereo display device context, the MGL does not automatically start stereo page flipping, and you must start this with a call to `MGL_startStereo`. You can also turn stereo mode on or off at any time (ie: you can turn it off when you go to your menu system) using the `MGL_stopStereo` and `MGL_startStereo` functions. Note that when stereo mode is disabled, the MGL always displays from the left eye buffer.

See Also

`MGL_createDisplayDC`, `MGL_destroyDC`, `MGL_startStereo`, `MGL_stopStereo`, `MGL_setBlueCodeIndex`

MGL_createWindowedDC

Create a new windowed device context.

Declaration

```
MGLDC * MGLAPI MGL_createWindowedDC (
    MGL_HWND hwnd)
```

Prototype In

mglwin.h

Parameters

hwnd Window handle with which to associate new device context

Return Value

Pointer to the allocated windowed device context, or NULL if not enough memory.

Description

Creates a new windowed device context for drawing information into a window on the Windows desktop. When you create a Windowed device context, you associate it with a standard Windows HWND for the window that you wish MGL to display its output on. Windowed device contexts are special device contexts in that you cannot directly access the surface for the device, nor can you actually use the MGL rasterizing functions to draw on the device surface. The only rasterizing functions supported are the *MGL_bitBlt* and *MGL_stretchBlt* for blt'ing data from memory device contexts to the window on the desktop.

However in order to change the color palette values for the data copied to the window, you must use the MGL palette functions on the windowed display device context. Note that MGL automatically takes care of creating a proper Windows identity palette for the windowed device context, so as long as you program the same palette values for the windowed device and the memory device you should get the maximum performance blt'ing speed.

See Also

MGL_createMemoryDC, *MGL_createDisplayDC*, *MGL_destroyDC*, *MGL_setWinDC*, *MGL_activatePalette*

MGL_defRect

Create a new rectangle.

Declaration

```
rect_t MGLAPI MGL_defRect(  
    int left,  
    int top,  
    int right,  
    int bottom)
```

Prototype In

mgraph.h

Parameters

<i>left</i>	Left coordinate of rectangle
<i>top</i>	Top coordinate of rectangle
<i>right</i>	Right coordinate of rectangle
<i>bottom</i>	Bottom coordinate of rectangle

Return Value

Newly created rectangle returned by value.

Description

Creates a new rectangle given a set of coordinates.

See Also

MGL_defRectPt

MGL_defRectPt

Create a new rectangle from two points

Declaration

```
rect_t MGLAPI MGL_defRectPt(  
    point_t leftTop,  
    point_t rightBottom)
```

Prototype In

mgraph.h

Parameters

<i>leftTop</i>	Upper left coordinate of rectangle
<i>rightBottom</i>	Lower right coordinate of rectangle

Return Value

Newly created rectangle returned by value.

Description

Creates a new rectangle given a set of points.

See Also

MGL_defRect

MGL_defaultAttributes

Reset all rasterizing attributes to their default values.

Declaration

```
void MGLAPI MGL_defaultAttributes(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc device context to be reset

Description

This function resets all of the device context attributes to their default values. Note that this function also sets the current viewport and clip rectangle for the device context to cover the entire device context surface.

See Also

MGL_getAttributes, MGL_restoreAttributes, MGL_getDefaultPalette

MGL_defaultColor

Returns the value for current default color (always white but value may vary).

Declaration

```
color_t MGLAPI MGL_defaultColor(void)
```

Prototype In

mgraph.h

Return Value

Default color value for current video mode (always white).

Description

Returns the default color value for the current video mode. This color value is white if the palette has not been changed, and will always be white in direct color modes. However, the numerical value for white will vary depending on the color depth.

See Also

MGL_setColor, *MGL_getColor*

MGL_destroyBuffer

Destroys an existing MGL buffer

Declaration

```
void MGLAPI MGL_destroyBuffer(  
    MGLBUF *buf)
```

Prototype In

mgraph.h

Parameters

buf MGL buffer to destroy

Description

This function destroys an MGL buffer, and frees all resources associated with the buffer.

See Also

MGL_createBuffer

MGL_destroyDC

Destroy a given device context.

Declaration

```
ibool MGLAPI MGL_destroyDC (  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context to be destroyed

Return Value

True if context was destroyed, false on error.

Description

Destroys a specified device context, freeing up all resources allocated by the device context. This can fail for a number of reasons, so check the *MGL_result* code to determine the cause of the failure.

If the device context that was destroyed was the last active display device context, the video mode is reset back to the original video mode (or back to the normal GDI desktop for Windows). Note that calling *MGL_exit* automatically destroys all currently allocated device contexts.

See Also

MGL_createDisplayDC, *MGL_createOffscreenDC*, *MGL_createMemoryDC*,
MGL_createWindowedDC

MGL_diffRegion

Compute the Boolean difference of two regions.

Declaration

```
ibool MGLAPI MGL_diffRegion(  
    region_t *r1,  
    const region_t *r2)
```

Prototype In

mgraph.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

MGL_diffRegionRect, MGL_unionRegion, MGL_sectRegion

MGL_diffRegionRect

Compute the Boolean difference of a region and a rectangle.

Declaration

```
ibool MGLAPI MGL_diffRegionRect(
    region_t *r1,
    const rect_t *r2)
```

Prototype In

mgraph.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

MGL_diffRegion, *MGL_unionRegion*, *MGL_sectRegion*

MGL_disableDriver

Disables a driver from being used in the detection process.

Declaration

```
int MGLAPI MGL_disableDriver(
    const char *name)
```

Prototype In

mgraph.h

Parameters

name Name of the driver to register

Return Value

grOK on success, error code on failure.

Description

This function disables a specific device driver so that it will no longer be used as part of the dynamic detection process for the currently active device. By default all device drivers are enabled, and this function allows the programmer to control the device detection process. For instance SciTech SNAP Graphics is used in preference to DirectDraw on Windows systems. To disable the SciTech SNAP Graphicsdriver and allow DirectDraw to be used, you would do the following:

```
MGL_disableDriver(MGL_SNAPNAME);
```

The names of the standard device drivers currently supported are:

<i>Driver</i>	Description
<i>MGL_VBENAME</i>	SciTech SNAP Graphics VGA and VESA VBE display driver
<i>MGL_SNAPNAME</i>	SciTech SNAP Graphics Accelerated display driver
<i>MGL_DDRAWNAME</i>	Microsoft DirectDraw accelerated display driver
<i>MGL_OPENGLNAME</i>	Microsoft hardware OpenGL display driver
<i>MGL_GLDIRECTNAME</i>	SciTech GLDirect hardware OpenGL display driver

See Also

MGL_enableAllDrivers, *MGL_enableAllOpenGLDrivers*

MGL_disjointRect

Determines if two rectangles are disjoint.

Declaration

```
ibool MGL_disjointRect(  
    rect_t r1,  
    rect_t r2)
```

Prototype In

mgraph.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

MGL_emptyRect, *MGL_equalRect*, *MGL_unionRect*, *MGL_sectRect*

MGL_divotSize

Number of bytes required to store a divot of specified size.

Declaration

```
long MGL_divotSize(  
    MGLDC *dc,  
    rect_t r)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to measure divot size from
<i>r</i>	Bounding rectangle of the divot area

Return Value

Size of the specified divot in bytes.

Description

This function is the same as *MGL_divotSizeCoord* however it takes entire rectangles as arguments instead of coordinates.

See Also

MGL_divotSizeCoord, *MGL_getDivot*, *MGL_putDivot*

MGL_divotSizeCoord

Number of bytes required to store a divot of specified size.

Declaration

```
long MGLAPI MGL_divotSizeCoord(
    MGLDC *dc,
    int left,
    int top,
    int right,
    int bottom)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to measure divot size from
<i>left</i>	Left coordinate of divot area
<i>top</i>	Top coordinate of divot area
<i>right</i>	Right coordinate of divot area
<i>bottom</i>	Bottom coordinate of divot area

Return Value

Size of the specified divot in bytes.

Description

Determines the number of bytes required to store a divot of the specified size taken from the current device context. A divot is a portion of video memory that needs to be temporarily saved and restored, such as implementing pull down menus and pop up dialog boxes. A divot must always be saved and restored to the same area, and will extend the dimensions of the area covered to obtain the maximum possible performance for saving and restoring the memory.

See Also

MGL_divotSize, *MGL_getDivot*, *MGL_putDivot*

MGL_doubleBuffer

Enables double buffering for the specified display device context.

Declaration

```
ibool MGLAPI MGL_doubleBuffer(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Return Value

True if double buffering is now enabled, false if not.

Description

Enables double buffered graphics mode for the specified device context if possible. When the device context is in double buffered mode, all active output is sent to the hidden backbuffer, while the current front buffer is being displayed. You then make calls to *MGL_swapBuffers* to swap the front and back buffers so that the previously hidden backbuffer is instantly displayed.

If you intend to start double buffered graphics, you should make sure you call the *MGL_createDisplayDC* function with the double buffer flag set to true, so that some of offscreen video memory will be allocated for the backbuffer. If the device context only has one video page available, double buffering cannot be started and this function will fail.

See Also

MGL_singleBuffer, *MGL_swapBuffers*

MGL_drawGlyph

Draws a monochrome glyph.

Declaration

```
void MGLAPI MGL_drawGlyph(  
    font_t *font,  
    int x,  
    int y,  
    uchar glyph)
```

Prototype In

mgraph.h

Parameters

<i>font</i>	Font containing the glyphs
<i>x</i>	x coordinate to draw glyph at
<i>y</i>	y coordinate to draw glyph at
<i>glyph</i>	Index of glyph to draw

Description

Rasterizes the specified monochrome glyph from the font file in the current color at the specified location. This is effectively the same as drawing a monochrome bitmap, but by storing all your monochrome bitmaps in a font file, the glyphs will be stored as efficiently as possible.

See Also

MGL_rotateGlyph, *MGL_mirrorGlyph*

MGL_drawRegion

Draw a solid complex region.

Declaration

```
void MGLAPI MGL_drawRegion(  
    int x,  
    int y,  
    const region_t *rgn)
```

Prototype In

mgraph.h

Parameters

<i>x</i>	x coordinate to draw region at
<i>y</i>	y coordinate to draw region at
<i>rgn</i>	region to draw

Description

Draws the complex region at the specified location in the current pattern and write mode.

See Also

MGL_newRegion, MGL_copyRegion, MGL_freeRegion, MGL_diffRegion, MGL_unionRegion, MGL_sectRegion

MGL_drawStr

Draws a text string at the current position.

Declaration

```
void MGLAPI MGL_drawStr(  
    const char *str)
```

Prototype In

mgraph.h

Parameters

str String to display

Description

Draws a string at the current position (CP) in the current drawing color, write mode, font, text direction and justification. The CP is moved so that drawing will begin directly after the end of the string, only if the horizontal justification is set to MGL_LEFT_TEXT, otherwise the CP is not moved.

See Also

MGL_drawStrXY, MGL_drawStrXY_W, MGL_drawStr_W, MGL_textHeight, MGL_textWidth, MGL_useFont

MGL_drawStrXY

Draws a text string at the specified position.

Declaration

```
void MGLAPI MGL_drawStrXY(  
    int x,  
    int y,  
    const char *str)
```

Prototype In

mgraph.h

Parameters

<i>x</i>	x coordinate to begin rasterizing the string at
<i>y</i>	y coordinate to begin rasterizing the string at
<i>str</i>	String to display

Description

Draws a string at the specified (x,y) position in the current drawing color, write mode, font, text direction and justification.

See Also

MGL_drawStrXY_W, *MGL_drawStr*, *MGL_drawStr_W*, *MGL_textHeight*, *MGL_textWidth*, *MGL_useFont*

MGL_drawStrXY_W

Draws a wide character string at the specified position.

Declaration

```
void MGLAPI MGL_drawStrXY_W(
    int x,
    int y,
    const wchar_t *str)
```

Prototype In

mgraph.h

Parameters

<i>x</i>	x coordinate to begin rasterizing the string at
<i>y</i>	y coordinate to begin rasterizing the string at
<i>str</i>	Wide character string to display

Description

Draws a string at the specified (x,y) position in the current drawing color, write mode, font, text direction and justification.

This function is the same as *MGL_drawStrXY*, but provides support for Unicode wide characters (for far-east languages).

Note: *Wide character fonts are only supported for bitmap and TrueType fonts. Vector fonts are not supported via this function.*

See Also

MGL_drawStrXY, *MGL_drawStr*, *MGL_drawStr_W*, *MGL_textHeight*, *MGL_textWidth*, *MGL_useFont*

MGL_drawStr_W

Draws a wide character string at the current position.

Declaration

```
void MGLAPI MGL_drawStr_W(
    const wchar_t *str)
```

Prototype In

mgraph.h

Parameters

str Wide character string to display

Description

Draws a string at the current position (CP) in the current drawing color, write mode, font, text direction and justification. The CP is moved so that drawing will begin directly after the end of the string, only if the horizontal justification is set to MGL_LEFT_TEXT, otherwise the CP is not moved.

This function is the same as *MGL_drawStr*, but provides support for Unicode wide characters (for far-east languages).

Note: *Wide character fonts are only supported for bitmap and TrueType fonts. Vector fonts are not supported via this function.*

See Also

MGL_drawStrXY, *MGL_drawStrXY_W*, *MGL_drawStr*, *MGL_textHeight*, *MGL_textWidth*, *MGL_useFont*

MGL_dstTransBlit

Copies a block of image data with destination transparency.

Declaration

```
void MGL_dstTransBlit(
    MGLDC *dst,
    MGLDC *src,
    rect_t srcRect,
    int dstLeft,
    int dstTop,
    color_t transparent,
    int op)
```

Prototype In

mgraph.h

Parameters

<i>dst</i>	Destination device context
<i>src</i>	Source device context
<i>srcRect</i>	Rectangle defining source image
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>transparent</i>	Transparent color to skip in source image
<i>op</i>	Write mode to use during Blt

Description

This function is the same as *MGL_dstTransBlitCoord*, however it takes a rectangle as a parameter instead of the four coordinates of a rectangle.

See Also

MGL_bitBlit, *MGL_bitBlitCoord*, *MGL_srcTransBlit*, *MGL_srcTransBlitCoord*,
MGL_dstTransBlit, *MGL_dstTransBlitCoord*, *MGL_bitBlitPatt*, *MGL_bitBlitPattCoord*,
MGL_bitBlitFx, *MGL_bitBlitFxCoord*, *MGL_stretchBlit*, *MGL_stretchBlitCoord*,
MGL_stretchBlitFx, *MGL_stretchBlitFxCoord*, *MGL_copyPage*, *MGL_copyPageCoord*

MGL_dstTransBlitCoord

Copies a block of image data with destination transparency.

Declaration

```
void MGLAPI MGL_dstTransBlitCoord(
    MGLDC *dst,
    MGLDC *src,
    int left,
    int top,
    int right,
    int bottom,
    int dstLeft,
    int dstTop,
    color_t transparent,
    int op)
```

Prototype In

mgraph.h

Parameters

<i>dst</i>	Destination device context
<i>src</i>	Source device context
<i>left</i>	Left coordinate of source image
<i>top</i>	Top coordinate of source image
<i>right</i>	Right coordinate of source image
<i>bottom</i>	Bottom coordinate of source image
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>transparent</i>	Transparent color to skip in source image
<i>op</i>	Write mode to use during Blt

Description

Copies a block of bitmap data from one device context to another with either source or destination transparency. When transferring the data with destination transparency, pixels in the destination image that are equal to the specified transparent color will be updated, and those pixels that are not the same will be skipped. This is effectively the operation performed for 'blueScreen'ing or color keying and can also be used for drawing transparent sprites. Note however that destination transparency is very slow in software compared to source transparency!

This routine has been highly optimized for maximum performance in all pixel depths, so will provide a very fast method for performing transparent sprite animation. However you may find that if you can use alternative techniques to pre-compile the sprites (like using run length encoding etc.) you will be able to build faster software based sprite animation code that can directly access the device context surface. However this routine can also be used to perform hardware accelerated Blt's between offscreen memory device's and the display device when running in fullscreen modes, providing the

hardware accelerator (if present) can support this operation. If you have a hardware accelerator capable of this, this will provide the ultimate performance for transparent sprite animation.

The source and destination rectangles are clipped according to the current clipping rectangles for the source and destination device contexts respectively.

Note: *If you are doing pixel format conversion at the same time (ie: color depth for source bitmap is different to the destination bitmap), then the transparent color value must be set to the translated destination pixel format. Ie: if you are blitting an 8bpp bitmap to a 32bpp device context, the transparent color must be a 32bpp value.*

Note: *This routine also only works with pixel depths that are at least 4 bits deep.*

See Also

MGL_bitBlit, MGL_bitBlitCoord, MGL_srcTransBlit, MGL_srcTransBlitCoord,
 MGL_dstTransBlit, MGL_dstTransBlitCoord, MGL_bitBlitPatt, MGL_bitBlitPattCoord,
 MGL_bitBlitFx, MGL_bitBlitFxCoord, MGL_stretchBlit, MGL_stretchBlitCoord,
 MGL_stretchBlitFx, MGL_stretchBlitFxCoord, MGL_copyPage, MGL_copyPageCoord

MGL_ellipse

Draws an ellipse outline.

Declaration

```
void MGLAPI MGL_ellipse(  
    rect_t extentRect)
```

Prototype In

mgraph.h

Parameters

extentRect Bounding rectangle for the ellipse

Description

Draws the outline of an ellipse given the bounding rectangle for the ellipse. The ellipse outline is drawn in the current pen color, style and size just inside the mathematical boundary of the bounding rectangle for the ellipse.

See Also

MGL_ellipseCoord, MGL_fillEllipse, MGL_ellipseArc, MGL_fillEllipseArc

MGL_ellipseArc

Draws an elliptical arc outline.

Declaration

```
void MGLAPI MGL_ellipseArc(
    rect_t extentRect,
    int startAngle,
    int endAngle)
```

Prototype In

mgraph.h

Parameters

<i>extentRect</i>	Bounding rectangle defining the arc (syntax 2)
<i>startAngle</i>	Starting angle for arc (in degrees)
<i>endAngle</i>	Ending angle for arc (in degrees)

Description

Draws the outline of an elliptical arc just inside the mathematical boundary of *extentRect*. *StartAngle* specifies where the arc begins and is treated MOD 360. *EndAngle* specifies where the arc ends and is also treated MOD 360. The angles are given in positive or negative degrees. Zero degrees is at 3 o'clock, 90 is at 12 o'clock high, 180 is at 9 o'clock and 270 is at 6 o'clock. Other angles are measured relative to the enclosing rectangle. Thus an angle of 45 degrees always defines a line from the center of the rectangle through it's top right corner, even if the rectangle isn't square. The ellipse outline is drawn in the current pen color, style and size.

See Also

MGL_ellipseArc, *MGL_fillEllipseArc*, *MGL_ellipse*, *MGL_fillEllipse*

MGL_ellipseArcCoord

Draws an elliptical arc outline.

Declaration

```
void MGLAPI MGL_ellipseArcCoord(
    int x,
    int y,
    int xradius,
    int yradius,
    int startAngle,
    int endAngle)
```

Prototype In

mgraph.h

Parameters

<i>x</i>	x coordinate for the center of elliptical arc (syntax 1)
<i>y</i>	y coordinate for the center of elliptical arc (syntax 1)
<i>xradius</i>	x radius for the elliptical arc (syntax 1)
<i>yradius</i>	y radius for the elliptical arc (syntax 1)
<i>startAngle</i>	Starting angle for arc (in degrees)
<i>endAngle</i>	Ending angle for arc (in degrees)

Description

Draws the outline of an elliptical given the center and radii for the ellipse. StartAngle specifies where the arc begins and is treated MOD 360. EndAngle specifies where the arc ends and is also treated MOD 360. The angles are given in positive or negative degrees. Zero degrees is at 3 o'clock, 90 is at 12 o'clock high, 180 is at 9 o'clock and 270 is at 6 o'clock. Other angles are measured relative to the enclosing rectangle. Thus an angle of 45 degrees always defines a line from the center of the rectangle through it's top right corner, even if the rectangle isn't square. The ellipse outline is drawn in the current pen color, style and size.

Note that this routine can only work with integer semi-major and semi-minor axes, but can sometimes be easier to work with (and is provided for compatibility with other graphics packages). The *MGL_ellipseArc* routine is more versatile than this, as you can have an ellipse with odd diameter values, which you cannot get with the this routine.

See Also

MGL_ellipseArc, *MGL_fillEllipseArc*, *MGL_ellipse*, *MGL_fillEllipse*

MGL_ellipseArcEngine

Generates the set of points on an elliptical arc.

Declaration

```
void MGLAPI MGL_ellipseArcEngine(
    rect_t extentRect,
    int startAngle,
    int endAngle,
    arc_coords_t *ac,
    void (MGLAPIP plotPoint)(
        long x,
        long y))
```

Prototype In

mgraph.h

Parameters

<i>extentRect</i>	Bounding rectangle defining the arc
<i>startAngle</i>	Starting angle for arc (in degrees)
<i>endAngle</i>	Ending angle for arc (in degrees)
<i>ac</i>	Place to store computed arc coordinates
<i>plotPoint</i>	Function to call for every point on the elliptical arc

Description

This routine generates the set of points on a elliptical arc, and is the same code used to generate elliptical arcs internally in MGL. You can call it to generate the set of points on an elliptical arc, calling your own plotPoint routine for every point on the arc. The points on the arc are rasterized in order from the starting angle to the ending angle. After the arc has been drawn, the arc coordinates are returned, which contains the actual center, starting and ending points for the arc.

See Also

MGL_ellipseEngine, MGL_lineEngine

MGL_ellipseCoord

Draws an ellipse outline.

Declaration

```
void MGLAPI MGL_ellipseCoord(
    int x,
    int y,
    int xradius,
    int yradius)
```

Prototype In

mgraph.h

Parameters

<i>x</i>	x coordinate for the center of ellipse (syntax 1)
<i>y</i>	y coordinate for the center of ellipse (syntax 1)
<i>xradius</i>	x radius for the ellipse (syntax 1)
<i>yradius</i>	y radius for the ellipse (syntax 1)

Description

Draws the outline of an ellipse given the center and radii for the ellipse. The ellipse outline is drawn in the current pen color, style and size just inside the mathematical boundary of the bounding rectangle for the ellipse.

Note that this routine can only work with integer semi-major and semi-minor axes, but can sometimes be easier to work with (and is provided for compatibility with other graphics libraries). The *MGL_ellipse* routine is more versatile than this, as you can have an ellipse with odd diameter values, which you cannot get with this routine.

See Also

MGL_ellipse, *MGL_fillEllipse*, *MGL_ellipseArc*, *MGL_fillEllipseArc*

MGL_ellipseEngine

Declaration

```
void MGLAPI MGL_ellipseEngine(
    rect_t extentRect,
    void (MGLAPIP setup) (
        int topY,
        int botY,
        int left,
        int right),
    void (MGLAPIP set4pixels) (
        ibool inc_x,
        ibool inc_y,
        ibool region1),
    void (MGLAPIP finished) (void))
```

Prototype In

mgraph.h

Parameters

<i>extentRect</i>	Bounding rectangle defining the ellipse
<i>setup</i>	Routine called to initialize pixel plotting routines
<i>set4pixels</i>	Routine called repeatedly for each set of 4 pixels
<i>finished</i>	Routine called to complete plotting pixels

Description

This routine generates the set of points on a ellipse, and is the same code used to generate ellipses internally in MGL. You can call it to generate the set of points on an ellipse, calling your own user defined plotting routines.

The setup routine is called before any pixels are plotted with the coordinates of the 4 seed points in the four ellipse quadrants.

The set4pixels routine is called repeatedly for each set of 4 pixels to be plotted, and specified whether the coordinates in the x and y directions should be incremented or remain the same. This state of the 4 pixel coordinates will need to be maintained by the user supplied routines.

The finished routine is called to clean up after generating all the points on the ellipse, such as releasing memory and rasterizing the ellipse if the rasterizing was deferred.

See Also

MGL_ellipseArcEngine, *MGL_lineEngine*

MGL_emptyRect

Determines if a rectangle is empty.

Declaration

```
ibool MGL_emptyRect(  
    rect_t r)
```

Prototype In

mgraph.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.

MGL_emptyRegion

Determines if a region is empty.

Declaration

```
ibool MGLAPI MGL_emptyRegion(  
    const region_t *r)
```

Prototype In

mgraph.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

MGL_equalRegion, MGL_unionRegion, MGL_diffRegion, MGL_sectRegion, MGL_offsetRegion, MGL_ptInRegion, MGL_ptInRegionCoord

MGL_enableAllDrivers

Enables all available non-OpenGL MGL device drivers for use.

Declaration

```
void MGLAPI MGL_enableAllDrivers(void)
```

Prototype In

mgraph.h

Description

This function enables all non-OpenGL specific MGL device drivers available on the target system for use, so that they will be used as part of the dynamic hardware detection process. This is normally the default case when you first initialise the MGL, so you only need to call this function if you have disabled any of the device drivers to modify the detection process. To enable OpenGL hardware support, use the *MGL_enableOpenGLDrivers* function.

See Also

MGL_enableOpenGLDrivers, *MGL_disableDriver*

MGL_enableOpenGLDrivers

Enables all available OpenGL specific MGL device drivers for use.

Declaration

```
void MGLAPI MGL_enableOpenGLDrivers(void)
```

Prototype In

mgraph.h

Description

This function enables all OpenGL specific MGL device drivers available on the target system for use, so that they will be used as part of the dynamic hardware detection process. If you wish to use OpenGL hardware acceleration in your application, you should call this function to register all the OpenGL specific drivers. OpenGL specific drivers are drivers that work with OS specific OpenGL hardware device drivers, and should only be used if the application is doing OpenGL rendering on the target OS. For non OpenGL applications, those drivers will be less efficient than the regular 2D only drivers.

Note: *This function needs to be called for **every** device in the system, since each device contains a separate list of enabled device drivers to use.*

See Also

MGL_enableAllDrivers, MGL_disableDriver

MGL_endDirectAccess

Disables direct framebuffer access.

Declaration

```
void MGLAPI MGL_endDirectAccess(void)
```

Prototype In

mgraph.h

Description

Disables direct framebuffer access so that you can use the accelerator functions to draw to the framebuffer memory. Note that calling this function is absolutely necessary when using hardware acceleration, as this function and the corresponding *MGL_beginDirectAccess* correctly arbitrate between the hardware accelerator graphics engine and your direct framebuffer writes.

See Also

MGL_beginDirectAccess, *MGL_endDirectAccess*, *MGL_beginDirectAccessDC*, *MGL_endDirectAccessDC*

MGL_endDirectAccessDC

Disables direct framebuffer access (device context specific)

Declaration

```
void MGLAPI MGL_endDirectAccessDC(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context to enable direct access to

Description

This is the same as *MGL_endDirectAccess* but it takes a device context pointer instead of requiring the device context to be the current device context.

See Also

MGL_beginDirectAccess, *MGL_endDirectAccess*, *MGL_beginDirectAccessDC*,
MGL_endDirectAccessDC

MGL_endPaint

Cleans up after a previous call to *MGL_beginPaint*.

Declaration

```
void MGLAPI MGL_endPaint(
    MGLDC *dc)
```

Prototype In

mglwin.h

Parameters

dc MGL windowed device context to use

Description

This function and its and the corresponding function *MGL_beginPaint()* should be called between the windows BeginPaint and EndPaint messages.

MGL_beginPaint() and *MGL_endPaint()* must bracket drawing functions that draw to a window type with a style of CS_PARENTDC or CS_CLASSDC. Such as dialog box controls. These types of windows allocate device handles on the fly so the HDC may change between calls to GetDC() or BeginPaint(). Therefore MGL cannot draw to these types of windows without knowing the new HDC after every BeginPaint() or GetDC() call.

A typical Windows WM_PAINT handler would be coded as follows:

```
case WM_PAINT:
    hdc = BeginPaint(hwnd, &ps);
    MGL_beginPaint(dc, hdc);
    // Do rasterizing code in here //
    MGL_bitBlt(dc, memDC, r, 0, 0, MGL_REPLACE_MODE);
    MGL_endPaint(dc);
    EndPaint(hwnd, &ps);
    return 0;
```

See Also

MGL_beginPaint

MGL_endPixel

Ends high speed pixel drawing operation.

Declaration

```
void MGLAPI MGL_endPixel(void)
```

Prototype In

mgraph.h

Description

This function ends a set of high speed pixel drawing operations, started with a call to *MGL_beginPixel*. This routine is intended primarily to ensure fast operation if you intend to plot more than a single pixel at a time.

See Also

MGL_beginPixel

MGL_enumerateFonts

Enumerates all available font families.

Declaration

```
void MGLAPI MGL_enumerateFonts(
    enumfntcallback_t callback,
    void *cookie)
typedef ibool (MGLAPIP enumfntcallback_t)(const font_info_t
*info, void *cookie)
```

Prototype In

mgraph.h

Parameters

<i>callback</i>	function that will be called for each font family
<i>cookie</i>	pointer to user data that will be passed to callback function

Description

This function finds all fonts in current directory and standard MGL locations and calls the callback function for every font family it has found. If the callback returns false, *MGL_enumerateFonts* immediately returns, even if it hasn't yet iterated over all available fonts.

The cookie argument is useful in multithreaded environment where it is necessary to distinguish between several concurrently running enumerations.

This function will scan directories for fonts when called for the first time. All subsequent calls are more efficient, because *MGL_enumerateFonts* will use font cache stored in memory.

MGL_enumerateFonts will attempt to create file named fntcache.inf in directories it scans for fonts. This file contains information about all fonts in the directory and will subsequently be used to further speed up fonts enumeration.

See Also

font_info_t

MGL_equalPoint

Compares two points to determine if they are equal.

Declaration

```
ibool MGL_equalPoint(  
    point_t p1,  
    point_t p2)
```

Prototype In

mgraph.h

Parameters

<i>p1</i>	The first point to compare.
<i>p2</i>	The second point to compare.

Return Value

True if the points are equal, false if they are not.

MGL_equalRect

Compares two rectangles for equality.

Declaration

```
ibool MGL_equalRect(  
    rect_t r1,  
    rect_t r2)
```

Prototype In

mgraph.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.

See Also

MGL_equalPoint, *MGL_equalRegion*

MGL_equalRegion

Determines if two regions are equal.

Declaration

```
ibool MGLAPI MGL_equalRegion(  
    const region_t *r1,  
    const region_t *r2)
```

Prototype In

mgraph.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

MGL_emptyRegion, *MGL_unionRegion*, *MGL_diffRegion*, *MGL_sectRegion*,
MGL_offsetRegion, *MGL_ptInRegion*, *MGL_ptInRegionCoord*

MGL_errorMsg

Returns a string describing an error condition code.

Declaration

```
const char * MGLAPI MGL_errorMsg(  
    int err)
```

Prototype In

mgraph.h

Parameters

err Error code to obtain string for

Return Value

Pointer to string describing the error condition.

Description

Returns a pointer to a string describing a specified error code. You can use this to convert the error codes from a numerical id return by *MGL_result* to a string which you can display for the users of your programs.

See Also

MGL_result

MGL_exit

Closes down the MGL.

Declaration

```
void MGLAPI MGL_exit(void)
```

Prototype In

mgraph.h

Description

This function closes down the MGL, deallocating any memory allocated for use by the MGL, and restoring the system back into the original display mode that was active before the MGL was started. This routine also properly removes all interrupt handlers and other system services that MGL hooked when it was initialized.

You must call this routine before you exit your application, to ensure that the system is properly terminated.

See Also

MGL_init

MGL_fadePalette

Fades the values for a color palette.

Declaration

```
ibool MGLAPI MGL_fadePalette(
    MGLDC *dc,
    palette_t *fullIntensity,
    int numColors,
    int startIndex,
    uchar intensity)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context with palette to fade
<i>fullIntensity</i>	Pointer to full intensity palette to fade from
<i>numColors</i>	Number of colors in palette to fade
<i>startIndex</i>	Starting index of first color to fade
<i>intensity</i>	Intensity for the final output palette (0 - 255)

Return Value

True if the entire output palette is black, false if not.

Description

This routine will take the values from a full intensity *palette_t* structure, fade the values and store them into a device context palette. The actual hardware palette will not be programmed at this stage, so you will then need to make a call to *MGL_realizePalette* to make the changes visible.

The intensity value is a number between 0 and 255 that defines the intensity of the output values. An intensity of 255 will produce the same output values as the input values. An intensity of 128 will product values in the output palette that are half the intensity of the input palette and an intensity of 0 produces an all black palette.

If the entire output palette is black, then the routine will return true, otherwise it will return false.

See Also

MGL_setPalette, *MGL_getPalette*, *MGL_rotatePalette*, *MGL_realizePalette*

MGL_fatalError

Declare a fatal error and exit gracefully.

Declaration

```
void MGLAPI MGL_fatalError(  
    const char *msg,  
    ...)
```

Prototype In

mgraph.h

Parameters

msg Message to display ... - Variable argument list to display

Description

A fatal internal error has occurred, so we shutdown the graphics systems, display the error message and quit. You should call this function to display your own internal fatal errors. You can use this function like printf(), allowing you to format the results for the output message, so long as the entire message string is less than 1024 bytes in length.

MGL_fclose

Closes an open disk file.

Declaration

```
int MGLAPI MGL_fclose(  
    FILE *f)
```

Prototype In

mgraph.h

Parameters

f Pointer to file to close

Return Value

0 on success, EOF on an error.

Description

This function is identical to the C library `fclose` function, but goes via MGL's internal file handling function pointers, which by default simply point to the standard C library functions. These functions are intended to allow the application programmer to override all the MGL file I/O functions with *MGL_setFileIO*, for custom I/O handling.

See Also

MGL_fopen, *MGL_fseek*, *MGL_ftell*, *MGL_fread*, *MGL_fwrite*, *MGL_setFileIO*

MGL_fillEllipse

Fills an ellipse.

Declaration

```
void MGLAPI MGL_fillEllipse(  
    rect_t extentRect)
```

Prototype In

mgraph.h

Description

Fills an ellipse given either the center and radii for the ellipse, or the bounding rectangle for the ellipse. The ellipse is filled in the current pen color and style just inside the mathematical boundary of the bounding rectangle for the ellipse.

Note that while this routine can only work with integer semi-major and semi-minor axes, it can sometimes be easier to work with (and is provided for compatibility with other graphics packages). *MGL_fillEllipseCoord* is a more versatile routine, as it allows ellipses with odd diameter values, which you cannot get with *MGL_fillEllipse*.

See Also

MGL_ellipse, *MGL_ellipseArc*, *MGL_fillEllipseArc*, *MGL_fillEllipseCoord*

MGL_fillEllipseArc

Fills an elliptical arc.

Declaration

```
void MGLAPI MGL_fillEllipseArc(
    rect_t extentRect,
    int startAngle,
    int endAngle)
```

Prototype In

mgraph.h

Parameters

<i>extentRect</i>	Bounding rectangle
<i>startAngle</i>	Starting angle for arc (in degrees)
<i>endAngle</i>	Ending angle for arc (in degrees)

Description

Fills an elliptical arc forming a wedge, just inside the mathematical boundary of *extentRect*. *StartAngle* specifies where the arc begins and is treated MOD 360. *EndAngle* specifies where the arc ends and is also treated MOD 360. The angles are given in positive or negative degrees. Zero degrees is at 3 o'clock, 90 is at 12 o'clock high, 180 is at 9 o'clock and 270 is at 6 o'clock. Other angles are measured relative to the enclosing rectangle. Thus an angle of 45 degrees always defines a line from the center of the rectangle through it's top right corner, even if the rectangle isn't square.

This routine is more versatile than *MGL_fillEllipseArcCoord*, as it allows an ellipse with odd diameter values, which you cannot get with the *MGL_fillEllipseArcCoord*.

See Also

MGL_fillEllipseArc, *MGL_ellipseArc*, *MGL_ellipse*, *MGL_fillEllipse*

MGL_fillEllipseArcCoord

Fills an elliptical arc.

Declaration

```
void MGLAPI MGL_fillEllipseArcCoord(
    int x,
    int y,
    int xradius,
    int yradius,
    int startAngle,
    int endAngle)
```

Prototype In

mgraph.h

Parameters

<i>x</i>	x coordinate for center of arc
<i>y</i>	y coordinate for center of arc
<i>xradius</i>	x radius for the arc
<i>yradius</i>	y radius for the arc
<i>startAngle</i>	Starting angle for arc (in degrees)
<i>endAngle</i>	Ending angle for arc (in degrees)

Description

Fills an elliptical arc given the center and radii for the ellipse. StartAngle specifies where the arc begins and is treated MOD 360. EndAngle specifies where the arc ends and is also treated MOD 360. The angles are given in positive or negative degrees. Zero degrees is at 3 o'clock, 90 is at 12 o'clock high, 180 is at 9 o'clock and 270 is at 6 o'clock. Other angles are measured relative to the enclosing rectangle. Thus an angle of 45 degrees always defines a line from the center of the rectangle through it's top right corner, even if the rectangle isn't square.

See Also

MGL_fillEllipseArc, MGL_ellipseArc, MGL_ellipse, MGL_fillEllipse

MGL_fillEllipseCoord

Fills an ellipse.

Declaration

```
void MGLAPI MGL_fillEllipseCoord(
    int x,
    int y,
    int xradius,
    int yradius)
```

Prototype In

mgraph.h

Parameters

<i>x</i>	x coordinate for center of ellipse (syntax 1)
<i>y</i>	y coordinate for center of ellipse (syntax 1)
<i>xradius</i>	x radius for the ellipse (syntax 1)
<i>yradius</i>	y radius for the ellipse (syntax 1)

Description

Fills an ellipse given either the center and radii for the ellipse, or the bounding rectangle for the ellipse. The ellipse is filled in the current pen color and style just inside the mathematical boundary of the bounding rectangle for the ellipse.

Note that the first routine can only work with integer semi-major and semi-minor axes, but can sometimes be easier to work with (and is provided for compatibility with other graphics packages). The second routine is more versatile than the first, as you can have an ellipse with odd diameter values, which you cannot get with the first routine.

See Also

MGL_ellipse, *MGL_ellipseArc*, *MGL_fillEllipseArc*

MGL_fillPolygon

Fills an arbitrary polygon.

Declaration

```
void MGLAPI MGL_fillPolygon(
    int count,
    point_t *vArray,
    int xOffset,
    int yOffset)
```

Prototype In

mgraph.h

Parameters

<i>count</i>	Number of vertices in polygon
<i>vArray</i>	Array of vertices in polygon
<i>xOffset</i>	x coordinate offset value
<i>yOffset</i>	y coordinate offset value

Description

This function is provided for backwards compatibility, and expects the array of points to be passed in as integers. Internally the MGL works in fixed point, so this function simply converts the coordinates to fixed point and passes them to the *MGL_fillPolygonFX* function. Hence *MGL_fillPolygonFX* is more efficient, so you should use that version instead.

Note: *All vertices are offset by (xOffset,yOffset).*

See Also

MGL_setPolygonType

MGL_fillPolygonCnvx

Scan converts a filled convex polygon.

Declaration

```
void MGLAPI MGL_fillPolygonCnvx(
    int count,
    point_t *vArray,
    int xOffset,
    int yOffset)
```

Prototype In

mgraph.h

Parameters

<i>count</i>	Number of vertices to draw
<i>vArray</i>	Array of vertices
<i>xOffset</i>	Offset of X coordinates
<i>yOffset</i>	Offset of Y coordinates

Description

This function is provided for backwards compatibility, and expects the array of points to be passed in as integers. Internally the MGL works in fixed point, so this function simply converts the coordinates to fixed point and passes them to the *MGL_fillPolygonCnvxFX* function. Hence *MGL_fillPolygonCnvxFX* is more efficient, so you should use that version instead.

Note: *All vertices are offset by (xOffset,yOffset).*

See Also

MGL_fillPolygonCnvxFX

MGL_fillPolygonCnvxFX

Scan converts a filled convex polygon.

Declaration

```
void MGLAPI MGL_fillPolygonCnvxFX(
    int count,
    fxpoint_t *vArray,
    int vinc,
    fix32_t xOffset,
    fix32_t yOffset)
```

Prototype In

mgraph.h

Parameters

<i>count</i>	Number of vertices to draw
<i>vArray</i>	Array of vertices
<i>vinc</i>	Increment to get to next vertex
<i>xOffset</i>	Offset of X coordinates
<i>yOffset</i>	Offset of Y coordinates

Description

A “convex” polygon is defined as a polygon such that every horizontal line drawn through the polygon would cross exactly two active edges (neither horizontal lines nor zero-length edges count as active edges; both are acceptable anywhere in the polygon). Right and left edges may cross (polygons may be nonsimple). Attempting to scan convert a polygon that does not fit this description will produce unpredictable results.

Note: All vertices are offset by (xOffset,yOffset).

See Also

MGL_fillPolygonFX

MGL_fillPolygonFX

Fills an arbitrary polygon.

Declaration

```
void MGLAPI MGL_fillPolygonFX(
    int count,
    fxpoint_t *vArray,
    int vinc,
    fix32_t xOffset,
    fix32_t yOffset)
```

Prototype In

mgraph.h

Parameters

<i>count</i>	Number of vertices in polygon
<i>vArray</i>	Array of vertices in polygon
<i>vinc</i>	Increment to get to next vertex in bytes
<i>xOffset</i>	x coordinate offset value
<i>yOffset</i>	y coordinate offset value

Description

These routines rasterize a filled arbitrary polygon in the current color and style. By default the routine will determine the type of the polygon being rasterized, and will rasterize convex polygons using a faster scan conversion routine, otherwise a general polygon scan conversion routine will be used. Thus you can rasterize any type of polygon that you desire.

A convex polygon is defined as a polygon such that every horizontal line drawn through the polygon would cross exactly two active edges (neither horizontal lines nor zero-length edges count as active edges; both are acceptable anywhere in the polygon). Right & left edges may cross (polygons may be non-simple).

Non-simple or self intersecting polygons will be rasterized using the standard in/out rule, where points are defined as being inside after crossing the first edge in the polygon, and then alternate between defined as inside then outside after crossing subsequent active edges in the polygon.

You may also use the *MGL_setPolygonType* routine to specify the type of polygons being rasterized. This may be *MGL_AUTO_POLYGON*, *MGL_CONVEX_POLYGON* or *MGL_COMPLEX_POLYGON*. Explicitly setting the polygon type will speed the drawing process.

As with all MGL polygon rasterizing routines, this routine does not rasterize the pixels down the right hand side or the bottom edges of the polygon. This ensures that pixels along shared edges of polygons are not rasterized twice, which can cause annoying pixel flashes in animation code. Note also that the edges in the polygon will always be

rasterized from top to bottom, to ensure that all shared edges will actually generate the same set of vertices, eliminating the possibility of pixel dropouts between shared edges in polygons.

Note: *All vertices are offset by (xOffset,yOffset).*

See Also

MGL_setPolygonType

MGL_fillRect

Draws a filled rectangle.

Declaration

```
void MGL_fillRect(  
    rect_t r)
```

Prototype In

mgraph.h

Parameters

r Rectangle to be filled

Description

This function is the same as *MGL_fillRectCoord*, however it takes an entire rectangle as the parameter instead of coordinates.

See Also

MGL_fillRectCoord, *MGL_fillRectPt*, *MGL_rect*

MGL_fillRectCoord

Draws a filled rectangle.

Declaration

```
void MGLAPI MGL_fillRectCoord(  
    int left,  
    int top,  
    int right,  
    int bottom)
```

Prototype In

mgraph.h

Parameters

<i>left</i>	Left coordinate of rectangle
<i>top</i>	Top coordinate of rectangle
<i>right</i>	Right coordinate of rectangle
<i>bottom</i>	Bottom coordinate of rectangle

Description

Fills a rectangle in the current drawing attributes. The mathematical definition of a rectangle does not include the right and bottom edges, so effectively the right and bottom edges are not rasterized (solving problems with shared edges).

See Also

MGL_fillRect, *MGL_fillRectPt*, *MGL_rect*

MGL_fillRectPt

Draws a filled rectangle.

Declaration

```
void MGL_fillRectPt(  
    point_t leftTop,  
    point_t rightBottom)
```

Prototype In

mgraph.h

Parameters

<i>leftTop</i>	Top left coordinate of rectangle
<i>rightBottom</i>	Bottom right coordinate of rectangle

Description

This function is the same as *MGL_fillRectCoord*, however it takes the top left and bottom right coordinates of the rectangle as two points instead of four coordinates.

See Also

MGL_fillRect, *MGL_fillRectPt*, *MGL_rect*

MGL_findMode

Finds the number of a mode given the resolution and color depth.

Declaration

```
int MGLAPI MGL_findMode(
    int xRes,
    int yRes,
    int bitsPerPixel)
```

Prototype In

mgraph.h

Parameters

<i>xRes</i>	Horizontal resolution for the display mode in pixels
<i>yRes</i>	Vertical resolution for the display mode in lines
<i>bitsPerPixel</i>	Color depth for the display mode

Return Value

MGL mode number for the mode, or -1 if not found.

Description

This function searches the list of available display modes in the MGL, looking for one that matches the specified resolution and color depth. This function is useful if your application always runs in a specific resolution and color depth, allowing you to quickly initialise the MGL without needing to write your own code to search the list of available modes.

Note: *If the hardware has not been detected when this call is made, the MGL will automatically detect the installed hardware the first time this function is called.*

See Also

MGL_init, MGL_availablePages, MGL_modeResolution, MGL_modeFlags, MGL_createDisplayDC

MGL_fopen

Opens a stream.

Declaration

```
FILE * MGLAPI MGL_fopen(  
    const char *filename,  
    const char *mode)
```

Prototype In

mgraph.h

Parameters

<i>filename</i>	Filename
<i>mode</i>	Mode to open file in.

Return Value

Pointer to newly opened stream, or NULL in the event of an error.

Description

This function is identical to the C library fopen function, but goes via the MGL's internal file handling function pointers, which by default simply points to the standard C library functions. These functions are intended to allow the application programmer to override all the MGL file I/O functions with *MGL_setFileIO*, for custom I/O handling.

See Also

MGL_fclose, *MGL_fseek*, *MGL_ftell*, *MGL_fread*, *MGL_fwrite*, *MGL_setFileIO*

MGL_fread

Reads data from a stream.

Declaration

```
size_t MGLAPI MGL_fread(
    void *ptr,
    size_t size,
    size_t n,
    FILE *f)
```

Prototype In

mgraph.h

Parameters

<i>ptr</i>	Pointer to block in stream at which to begin read
<i>size</i>	Size of items to be read from stream
<i>n</i>	Number of items to be read from stream
<i>f</i>	Stream to be read

Return Value

Number of items read in, or a short count (possibly 0).

Description

This function is identical to the C library fread function, but goes via the MGL's internal file handling function pointers, which by default simply points to the standard C library functions. These functions are intended to allow the application programmer to override all the MGL file I/O functions with *MGL_setFileIO*, for custom I/O handling.

See Also

MGL_fopen, *MGL_fclose*, *MGL_fseek*, *MGL_ftell*, *MGL_fwrite*, *MGL_setFileIO*

MGL_freeRegion

Frees all the memory allocated by the complex region.

Declaration

```
void MGLAPI MGL_freeRegion(  
    region_t *r)
```

Prototype In

mgraph.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

MGL_newRegion, *MGL_copyRegion*

MGL_fseek

Repositions the file pointer on a stream.

Declaration

```
int MGLAPI MGL_fseek(  
    FILE *f,  
    long offset,  
    int whence)
```

Prototype In

mgraph.h

Parameters

<i>f</i>	Stream of interest
<i>offset</i>	Offset of location from whence
<i>whence</i>	New location of file pointer

Return Value

0 if move was successful, otherwise non-zero.

Description

This function is identical to the C library `fseek` function, but goes via MGL's internal file handling function pointers, which by default simply point to the standard C library functions. These functions are intended to allow the application programmer to override all the MGL file I/O functions with *MGL_setFileIO*, for custom I/O handling.

See Also

MGL_fopen, *MGL_fclose*, *MGL_ftell*, *MGL_fread*, *MGL_fwrite*, *MGL_setFileIO*

MGL_ftell

Returns the current file pointer.

Declaration

```
long MGLAPI MGL_ftell(  
    FILE *f)
```

Prototype In

mgraph.h

Parameters

f Pointer to file of interest

Return Value

Current file pointer on success, -1L on error.

Description

This function is identical to the C library `fopen` function, but goes via the MGL's internal file handling function pointers, which by default simply points to the standard C library functions. These functions are intended to allow the application programmer to override all the MGL file I/O functions with *MGL_setFileIO*, for custom I/O handling.

See Also

MGL_fopen, *MGL_fclose*, *MGL_fseek*, *MGL_fread*, *MGL_fwrite*, *MGL_setFileIO*

MGL_fwrite

Writes to a stream.

Declaration

```
size_t MGLAPI MGL_fwrite(  
    const void *ptr,  
    size_t size,  
    size_t n,  
    FILE *f)
```

Prototype In

mgraph.h

Parameters

<i>ptr</i>	Pointer to the starting location of data to be written
<i>size</i>	Size of items to be written to file
<i>n</i>	Number of items to be written to file
<i>f</i>	Pointer to the file stream to write the data to

Return Value

The number of items written.

Description

This function is identical to the C library `fopen` function, but goes via the MGL's internal file handling function pointers, which by default simply points to the standard C library functions. These functions are intended to allow the application programmer to override all the MGL file I/O functions with *MGL_setFileIO*, for custom I/O handling.

See Also

MGL_fopen, *MGL_fclos*, *MGL_fseek*, *MGL_ftell*, *MGL_fread*, *MGL_setFileIO*

MGL_getActivePage

Returns the currently active hardware display page.

Declaration

```
int MGLAPI MGL_getActivePage(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context of interest

Return Value

Currently active hardware display page.

Description

This function returns the currently active hardware display page number. The first hardware display page is 0, the second is 1 and so on. The number of available hardware pages depends on the type of underlying hardware, the video mode resolution and amount of video memory installed. Thus not all video modes support multiple hardware display pages.

All MGL output is always sent to the currently active hardware display page, and changing the active and visual display pages is used to implement double buffering.

See Also

MGL_setActivePage, *MGL_getVisualPage*, *MGL_setVisualPage*.

MGL_getAlphaValue

Returns the current constant alpha value for blending functions.

Declaration

```
uchar MGLAPI MGL_getAlphaValue(void)
```

Prototype In

mgraph.h

Return Value

Current constant alpha value for blending functions.

Description

This function returns the current constant alpha value for pixel blending function in the MGL.

See Also

MGL_setSrcBlendFunc, MGL_setDstBlendFunc, *MGL_setAlphaValue*

MGL_getArcCoords

Returns the starting and ending arc coordinates.

Declaration

```
void MGLAPI MGL_getArcCoords(  
    arc_coords_t *coords)
```

Prototype In

mgraph.h

Parameters

coords Pointer to structure to store coordinates

Description

This function returns the center coordinate, and starting and ending points on the ellipse that define the last elliptical arc that was rasterized. You can then use these coordinates to draw a line from the center of the ellipse to the starting and ending points to complete the outline of an elliptical wedge.

Note that you must call this routine immediately after calling the *MGL_ellipseArc* family of routines.

See Also

MGL_ellipseArc, *MGL_fillEllipseArc*

MGL_getAspectRatio

Returns the current device context aspect ratio.

Declaration

```
int MGLAPI MGL_getAspectRatio(void)
```

Prototype In

mgraph.h

Return Value

Current video mode aspect ratio * 1000.

Description

This function returns the aspect ratio of the currently active output device's physical pixels. This ratio is equal to:

$$\frac{\text{pixel x size}}{\text{pixel y size}} \times 1000$$

The device context aspect ratio can be used to display circles and squares on the device by approximating them with ellipses and rectangles of the appropriate dimensions. Thus in order to determine the number of pixels in the y direction for a square with 100 pixels in the x direction, we can simply use the code:

```
y_pixels = ((long)x_pixels * 1000) / aspectratio
```

Note the cast to a long to avoid arithmetic overflow, as the aspect ratio is returned as an integer value with 1000 being a 1:1 aspect ratio.

See Also

MGL_setAspectRatio

MGL_getAttributes

Returns a copy of the current rasterizing attributes.

Declaration

```
void MGLAPI MGL_getAttributes(  
    attributes_t *attr)
```

Prototype In

mgraph.h

Parameters

attr Pointer to structure to store attribute values in

Description

This function returns a copy of the currently active attributes. You can use this routine to save the state of MGL and later restore this state with the *MGL_restoreAttributes* routine.

See Also

MGL_restoreAttributes

MGL_getBackColor

Returns the current background color value.

Declaration

```
color_t MGLAPI MGL_getBackColor(void)
```

Prototype In

mgraph.h

Return Value

Current background color value.

Description

Returns the current background color value. The background color value is used to clear the display and viewport with the *MGL_clearDevice* and *MGL_clearViewport* routines, and is also used for filling solid primitives in the MGL_BITMAP_OPAQUE fill mode.

See Also

MGL_setBackColor, *MGL_getColor*, *MGL_setColor*

MGL_getBackMode

Returns the current background mode for monochrome bitmap rendering.

Declaration

```
int MGLAPI MGL_getBackMode(void)
```

Prototype In

mgraph.h

Return Value

Current background mode.

Description

This function returns the current background mode for the device context. Defined modes are enumerated in *MGL_backModes*.

See Also

MGL_setBackMode

MGL_getBitmapFromDC

Copy a portion of a device context as a lightweight memory bitmap.

Declaration

```
bitmap_t * MGLAPI MGL_getBitmapFromDC (
    MGLDC *dc,
    int left,
    int top,
    int right,
    int bottom,
    ibool savePalette)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to save from
<i>left</i>	Left coordinate of bitmap to save
<i>top</i>	Top coordinate of bitmap to save
<i>right</i>	Right coordinate of bitmap to save
<i>bottom</i>	Bottom coordinate of bitmap to save
<i>savePalette</i>	Save palette with bitmap.

Return Value

Pointer to allocated bitmap, NULL on error.

Description

This function copies a portion of a device context as a lightweight memory bitmap. If this function fails (for instance if out of memory), it will return NULL and you can get the error code from the *MGL_result* function. Otherwise this function will return a pointer to a new lightweight bitmap containing the bitmap data.

Note that the source rectangle for the bitmap to be saved is not clipped to the current clipping rectangle for the device context, but it is mapped to the current viewport. If you specify dimensions that are larger than the current device context, you will get garbage in the bitmap file as a result.

See Also

MGL_loadBitmap, *MGL_saveBitmapFromDC*

MGL_getBitmapSize

Obtain the dimensions of a bitmap file from disk.

Declaration

```
ibool MGLAPI MGL_getBitmapSize(
    const char *bitmapName,
    int *width,
    int *height,
    int *bitsPerPixel,
    pixel_format_t *pf)
```

Prototype In

mgraph.h

Parameters

<i>bitmapName</i>	Name of the bitmap file to load header for
<i>width</i>	Place to store the bitmap width
<i>height</i>	Place to store the bitmap height
<i>bitsPerPixel</i>	Place to store the bitmap pixel depth
<i>pf</i>	Place to store the bitmap pixel format information

Return Value

True if the bitmap was found, false if not.

Description

This functions loads all the header information for a bitmap file from disk, without actually loading the bits for the bitmap surface. This is useful to determine the dimensions and pixel format for the bitmap before it is loaded, so you can create an appropriate memory device context that you can load the bitmap into with the *MGL_loadBitmapIntoDC* function.

See Also

MGL_loadBitmap, *MGL_loadBitmapExt*

MGL_getBitmapSizeExt

Obtain the dimensions of a bitmap from an open file.

Declaration

```
ibool MGLAPI MGL_getBitmapSizeExt(
    FILE *f,
    ulong dwOffset,
    ulong dwSize,
    int *width,
    int *height,
    int *bitsPerPixel,
    pixel_format_t *pf)
```

Prototype In

mgraph.h

Parameters

<i>f</i>	Pointer to opened bitmap file
<i>dwOffset</i>	Offset into the file
<i>dwSize</i>	Size of the bitmap file
<i>width</i>	Place to store the bitmap width
<i>height</i>	Place to store the bitmap height
<i>bitsPerPixel</i>	Place to store the bitmap pixel depth
<i>pf</i>	Place to store the bitmap pixel format

Return Value

True if the bitmap was found, otherwise false.

Description

This function is the same as *MGL_getBitmapSize*, however it works with a previously opened file. This allows you to create your own large files with multiple files embedded in them.

See Also

MGL_getBitmapSize

MGL_getBitsPerPixel

Returns the pixel depth for the device context.

Declaration

```
int MGLAPI MGL_getBitsPerPixel(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Pointer to device context of interest

Return Value

Pixel depth for the device context.

See Also

MGL_getPixelFormat

MGL_getBlendFunc

Returns the current source and destination pixel blending functions.

Declaration

```
void MGLAPI MGL_getBlendFunc(  
    int *srcBlendFunc,  
    int *dstBlendFunc)
```

Prototype In

mgraph.h

Parameters

<i>srcBlendFunc</i>	Place to store source blending function
<i>dstBlendFunc</i>	Place to store destination blending function

Description

This function returns the current source and destination pixel blending functions used for pixel blending in the MGL. By default pixel blending is disabled, and the blending function is set to MGL_BLEND_NONE. For a more detailed description of the blending functions, see the documentation for the *MGL_blendFuncType* enumeration.

See Also

MGL_setBlendFunc, *MGL_setAlphaValue*

MGL_getCP

Returns the current position value.

Declaration

```
void MGLAPI MGL_getCP(  
    point_t *CP)
```

Prototype In

mgraph.h

Parameters

CP Place to store the current position

Description

Returns the current position (CP). The CP is the current logical graphics cursor position, and is used by a number of routines to determine where to begin drawing output. You can use the *MGL_moveTo* routine to directly move the CP to a new position.

See Also

MGL_moveTo, *MGL_moveRel*, *MGL_lineTo*, *MGL_lineRel*, *MGL_drawStr*

MGL_getCharMetrics

Computes the character metrics for a specific character.

Declaration

```
void MGLAPI MGL_getCharMetrics(  
    char ch,  
    metrics_t *metrics)
```

Prototype In

mgraph.h

Parameters

<i>ch</i>	Character to measure
<i>metrics</i>	Place to store the resulting metrics

Description

This function computes the character metrics for a specific character. The character metrics define specific characters width, height, ascent, descent and other values. These values can then be used to correctly position the character with pixel precise positioning.

All values are defined in pixels and will be as accurate as possible given the current fonts scaling factor (only vector fonts can be scaled).

See Also

MGL_getFontMetrics, *MGL_getCharMetrics_W*

MGL_getCharMetrics_W

Computes the character metrics for a specific wide character.

Declaration

```
void MGLAPI MGL_getCharMetrics_W(  
    wchar_t ch,  
    metrics_t *metrics)
```

Prototype In

mgraph.h

Parameters

<i>ch</i>	Wide character to measure
<i>metrics</i>	Place to store the resulting metrics

Description

This function computes the character metrics for a specific wide character. The character metrics define specific characters width, height, ascent, descent and other values. These values can then be used to correctly position the character with pixel precise positioning. This function is the same as *MGL_getCharMetrics*, but provides support for Unicode wide characters (for far-east languages).

All values are defined in pixels and will be as accurate as possible given the current fonts scaling factor (only vector fonts can be scaled).

See Also

MGL_getFontMetrics, *MGL_getCharMetrics*

MGL_getClipRect

Returns the current clipping rectangle.

Declaration

```
void MGLAPI MGL_getClipRect(  
    rect_t *clip)
```

Prototype In

mgraph.h

Parameters

clip Place to store the current clipping rectangle

Description

Returns the current clipping rectangle coordinates. The current clipping rectangle is used to clip all output, and is always defined as being relative to the currently active viewport. The clipping rectangle can be no larger than the currently active viewport.

Note: *If a complex clip region is currently active, this function returns the bounding rectangle for the complex clip region.*

See Also

MGL_getClipRectDC, MGL_setClipRect, MGL_getClipRegion

MGL_getClipRectDC

Returns the current clipping rectangle for a specific DC.

Declaration

```
void MGLAPI MGL_getClipRectDC(
    MGLDC *dc,
    rect_t *clip)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Display device context in which the rectangle is defined
<i>clip</i>	Place to store the current clipping rectangle

Description

This function is the same as *MGL_getClipRect*, however the device context does not have to be the current device context.

Note: *If a complex clip region is currently active, this function returns the bounding rectangle for the complex clip region.*

See Also

MGL_getClipRect, MGL_setClipRect, MGL_getClipRegion

MGL_getClipRegion

Returns the current complex clipping region.

Declaration

```
void MGLAPI MGL_getClipRegion(  
    region_t *region)
```

Prototype In

mgraph.h

Parameters

region Place to store the current complex clipping region

Description

Returns the current complex clipping region for the current device context. This function works even if no complex clip region is active, and only a simple clip rectangle is in use. In this case, the resulting clip region is a simple region that contains only the clip rectangle for the device context.

See Also

MGL_getClipRegionDC, MGL_setClipRegion, MGL_setClipRect

MGL_getClipRegionDC

Returns the current complex clipping region for a specific DC.

Declaration

```
void MGLAPI MGL_getClipRegionDC(  
    MGLDC *dc,  
    region_t *region)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to get clip region for
<i>region</i>	Place to store the current complex clipping region

Description

Returns the current complex clipping region for the specific device context. This function works even if no complex clip region is active, and only a simple clip rectangle is in use. In this case, the resulting clip region is a simple region that contains only the clip rectangle for the device context.

See Also

MGL_getClipRegion, MGL_setClipRegion, MGL_setClipRect

MGL_getColor

Returns the current foreground color.

Declaration

```
color_t MGLAPI MGL_getColor(void)
```

Prototype In

mgraph.h

Return Value

Current foreground color.

See Also

MGL_setColor, MGL_getBackColor, MGL_setBackColor

MGL_getCurrentScanLine

Returns the line currently being displayed on the screen.

Declaration

```
int MGLAPI MGL_getCurrentScanLine(
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Display device context of interest

Return Value

Line currently being display on the screen, -1 if not available.

Description

This function will determine what line the hardware is currently display on the screen (the current raster scan), and return this value. It will be a value in the range from 0 to the vertical total value (usually about 10% more than the vertical number of lines in the mode). This can be useful for implementing raster scan chasing algorithms for smooth animation when fullscreen page flipping is not available.

Note: *This function may not be available on all display devices, so make sure you check the return value to see if the function is implemented and available. If the function is not available, this function will return a value of -1.*

See Also

MGL_vSync, MGL_isVSync

MGL_getDefaultPalette

Returns the default palette for the device.

Declaration

```
void MGLAPI MGL_getDefaultPalette(  
    MGLDC *dc,  
    palette_t *pal)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context of interest
<i>pal</i>	Place to store the default palette values

Description

Copies the default palette for the specified device context into the passed palette structure.

Note: *The size of the default palette can be found with a call to MGL_getPaletteSize().*

See Also

MGL_setPalette, MGL_getPalette

MGL_getDisplayStart

Returns the current hardware display starting coordinates.

Declaration

```
void MGLAPI MGL_getDisplayStart(  
    MGLDC *dc,  
    int *x,  
    int *y)
```

Prototype In mgraph.h

Parameters

<i>dc</i>	Hardware scrolling device context
<i>x</i>	Place to store the display start x coordinate
<i>y</i>	Place to store the display start y coordinate

Description

This function returns the current hardware display start coordinates for the hardware scrolling display device context. You can change the display start address with the *MGL_setDisplayStart* function.

See Also

MGL_setDisplayStart

MGL_getDitherMode

Returns the current dithering mode for all blitting operations.

Declaration

```
int MGLAPI MGL_getDitherMode(void)
```

Prototype In

mgraph.h

Return Value

Current dithering mode mask.

Description

This function returns the current dithering mode used when blitting RGB images to 8, 15 and 16 bits per pixel device context.

See Also

MGL_setDitherMode

MGL_getDivot

Saves a divot of video memory into system RAM.

Declaration

```
void MGL_getDivot(  
    MGLDC dc,  
    rect_t,  
    void *divot)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to save divot from
<i>divot</i>	Pointer to area to store the video memory in

Description

This function is the same as *MGL_getDivotCoord* however it takes entire rectangles as arguments instead of coordinates.

See Also

MGL_putDivot, *MGL_divotSize*, *PM_malloc*

MGL_getDivotCoord

Saves a divot of video memory into system RAM.

Declaration

```
void MGLAPI MGL_getDivotCoord(
    MGLDC *dc,
    int left,
    int top,
    int right,
    int bottom,
    void *divot)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to save divot from
<i>left</i>	Left coordinate of area to save
<i>top</i>	Top coordinate of area to save
<i>right</i>	Right coordinate of area to save
<i>bottom</i>	Bottom coordinate of area to save
<i>divot</i>	Pointer to area to store the video memory in

Description

This function copies a block of video memory from the active page of the current device context into a system RAM buffer. A divot is defined as being a rectangular area of video memory that you wish to save, however the bounding rectangle for the divot is expanded slightly to properly aligned boundaries for the absolute maximum performance with the current device context. This function is generally used to store the video memory behind pull down menus and pop up dialog boxes, and the memory can only be restored to exactly the same position that it was saved from.

You must pre-allocate enough space to hold the entire divot in system RAM. Use the *MGL_divotSize* routine to determine the size of the memory block required to store the divot.

See Also

MGL_getDivot, *MGL_putDivot*, *MGL_divotSize*, *PM_malloc*

MGL_getDotsPerInch

Gets the outline font dots per inch value.

Declaration

```
void MGLAPI MGL_getDotsPerInch(  
    int *xDPI,  
    int *yDPI)
```

Prototype In

mgraph.h

Parameters

<i>xDPI</i>	Place to store horizontal dots per inch
<i>yDPI</i>	Place to store vertical dots per inch

Description

This function returns the current the horizontal and vertical dots per inch factor, which is used to generate outline font bitmap glyphs. Making the value larger results in larger font bitmaps. The default value for PC compatible systems is 96 DPI.

MGL_getFont

Returns the currently active font.

Declaration

```
font_t * MGLAPI MGL_getFont(void)
```

Prototype In

mgraph.h

Return Value

Pointer to currently active font.

Description

Returns a pointer to the currently active font. The currently active font is used to perform all text output by MGL.

See Also

MGL_useFont, MGL_loadFont, MGL_unloadFont

MGL_getFontAntiAliasPalette

Returns the font anti-aliasing palette for color index modes.

Declaration

```
void MGLAPI MGL_getFontAntiAliasPalette(
    color_t *colorfg,
    color_t *color75,
    color_t *color50,
    color_t *color25,
    color_t *colorbg)
```

Prototype In

mgraph.h

Parameters

<i>colorfg</i>	Color used to represent 100% of foreground
<i>color75</i>	Color used to represent 75% blend of foreground 25% background.
<i>color50</i>	Color used to represent 50% blend of foreground 50% background.
<i>color25</i>	Color used to represent 25% blend of foreground 75% background.
<i>colorbg</i>	Color used to represent 100% of background.

Description

This function returns the values of the anti-aliasing palette used to draw anti-aliased fonts when in non-blended mode in color index display modes.

See Also

MGL_setFontBlendMode, *MGL_setFontAntiAliasPalette*

MGL_getFontBlendMode

Returns the blending mode for anti-aliased fonts.

Declaration

```
int MGLAPI MGL_getFontBlendMode(void)
```

Prototype In

mgraph.h

Return Value

Current font blending mode, enumerated by *MGL_fontBlendType*.

Description

This function returns the current font blending mode used to render anti-aliased fonts.

See Also

MGL_setFontBlendMode, *MGL_setFontAntiAliasPalette*

MGL_getFontMetrics

Returns the currently active font metrics.

Declaration

```
void MGLAPI MGL_getFontMetrics(  
    metrics_t *m)
```

Prototype In

mgraph.h

Description

This function computes the font metrics for the current font. The metrics are computed in pixels and will be as accurate as possible given the current font's scaling factor (only vector fonts can be scaled however).

See Also

MGL_getCharMetrics

MGL_getFullScreenWindow

Returns the current fullscreen window handle.

Declaration

```
MGL_HWND MGLAPI MGL_getFullScreenWindow(void)
```

Prototype In

mglwin.h

Return Value

Current fullscreen window handle.

Description

This function returns the handle to the current fullscreen window. When you are running in fullscreen modes under Windows, MGL always maintains a fullscreen, topmost window that is used for event handling.

If you are using the DirectSound libraries for sound output, you will need to inform DirectSound what your fullscreen window is so that it can correctly mute the sound for your application when the focus is lost to another application. If you do not do this, when you switch to fullscreen modes all sound output via DirectSound will be muted (assuming you are requesting exclusive mode).

See Also

MGL_registerEventProc

MGL_getGammaRamp

Returns the currently active hardware gamma correction ramp.

Declaration

```
ibool MGLAPI MGL_getGammaRamp(
    MGLDC *dc,
    palette_ext_t *gamma,
    int num,
    int index)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context of interest
<i>gamma</i>	Place to store the retrieved values
<i>num</i>	Number of gamma entries to retrieve
<i>index</i>	Starting gamma entry index to retrieve

Description

This function copies part or all of the currently active hardware gamma correction ramp and stores it in the gamma array. You can specify only a subset of the gamma entries to be retrieved with the startIndex and numColors arguments.

See Also

MGL_setGammaRamp

MGL_getGlyphHeight

Returns the height of the glyphs in the specified font.

Declaration

```
int MGLAPI MGL_getGlyphHeight(  
    font_t *font)
```

Prototype In

mgraph.h

Parameters

font Font of interest

Return Value

Height of all the glyphs in the specified font.

See Also

MGL_getGlyphWidth

MGL_getGlyphWidth

Returns the width of a specified glyph in the specified font.

Declaration

```
int MGLAPI MGL_getGlyphWidth(  
    font_t *font,  
    uchar glyph)
```

Prototype In

mgraph.h

Parameters

<i>font</i>	Font of interest
<i>glyph</i>	Index of glyph in font file to measure

Return Value

Returns the width of the specified glyph.

See Also

MGL_getGlyphHeight

MGL_getHalfTonePalette

Returns a copy of the MGL halftone palette.

Declaration

```
void MGLAPI MGL_getHalfTonePalette(  
    palette_t *pal)
```

Prototype In

mgraph.h

Parameters

pal Place to store the halftone palette values

Description

This function copies the MGL halftone palette into the specified palette structure. The halftone palette always contains 256 colors, and hence the palette array must contain 256 palette entries. This palette is a special palette used by MGL when running in RGB dithered rasterizing mode for 8 bit video modes. If you intend to enable RGB dithering with the MGL_setColorMapMode function, you must get a copy of the halftone palette and program the hardware palette for your display device or windowed device to be the same as this halftone palette.

Note that the MGL halftone palette is compatible with the standard Windows halftone palette, so you can perform 8 bit dithering operations in a window without needing to go into SYSPAL_STATIC mode.

See Also

MGL_setPalette, *MGL_realizePalette*, *MGL_setColorMapMode*

MGL_getHardwareFlags

Returns the current hardware acceleration flags for the display device context.

Declaration

```
long MGLAPI MGL_getHardwareFlags(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context of interest

Return Value

Current hardware acceleration flags for the device context.

Description

This function returns the current hardware acceleration flags for the display device context (this function returns 0 for non-display device contexts). The set of hardware acceleration flags inform the application of what specific hardware acceleration features the underlying video hardware has, so that the application can tailor its use of specific MGL functions. For instance an application may check if the hardware has transparent BitBlt capabilities for sprite animation, and if not will use its own application specific set of routines that rasterize directly to the display surface rather than using the MGL specific functions.

The set of hardware acceleration flags that are returned will be a logical combination of one or more of the values enumerated in *MGL_hardwareFlagsType*.

MGL_getJPEGSize

Obtain the dimensions of a JPEG file from disk.

Declaration

```
ibool MGLAPI MGL_getJPEGSize(
    const char *JPEGName,
    int *width,
    int *height,
    int *bitsPerPixel,
    pixel_format_t *pf)
```

Prototype In

mgraph.h

Parameters

<i>JPEGName</i>	Name of the bitmap file to load header for
<i>width</i>	Place to store the bitmap width
<i>height</i>	Place to store the bitmap height
<i>bitsPerPixel</i>	Place to store the bitmap pixel depth
<i>pf</i>	Place to store the bitmap pixel format information

Return Value

True if the JPEG file was found, false if not.

Description

This functions loads all the header information for a JPEG file from disk, without actually loading the bits for the bitmap surface. This is useful to determine the dimensions and pixel format for the bitmap before it is loaded, so you can create an appropriate memory device context that you can load the bitmap into with the *MGL_loadJPEGIntoDC* function.

Note that JPEG files are inherently 24-bit, so when you call this function it will always return information for a 24-bit RGB pixel format image.

See Also

MGL_loadJPEG, *MGL_loadJPEGIntoDC*

MGL_getJPEGSizeExt

Obtain the dimensions of a JPEG file from an opened file.

Declaration

```
ibool MGLAPI MGL_getJPEGSizeExt(
    FILE *f,
    ulong dwOffset,
    ulong dwSize,
    int *width,
    int *height,
    int *bitsPerPixel,
    pixel_format_t *pf)
```

Prototype In

mgraph.h

Parameters

<i>f</i>	Open binary file to read bitmap from
<i>dwOffset</i>	Offset to start of JPEG file
<i>dwSize</i>	Size of JPEG file
<i>width</i>	Width of bitmap
<i>height</i>	Height of bitmap
<i>bitsPerPixel</i>	Pixel depth of bitmap
<i>pf</i>	Place to store the bitmap pixel format information

Return Value

True if JPEG was found, false if not.

Description

This function is the same as *MGL_getJPEGSize*, however it works with a previously opened file. This allows you to create your own large files with multiple files embedded in them.

Note that JPEG files are inherently 24-bit, so when you call this function it will always return information for a 24-bit RGB pixel format image.

See Also

MGL_getJPEGSize

MGL_getLineStipple

Returns the current line stipple pattern.

Declaration

```
ushort MGLAPI MGL_getLineStipple(void)
```

Prototype In

mgraph.h

Description

Return the current line stipple pattern. Refer to *MGL_setLineStipple* for more information on stippled lines.

See Also

MGL_setLineStyle, *MGL_setLineStipple*, *MGL_setLineStippleCount*

MGL_getLineStippleCount

Returns the current line stipple counter.

Declaration

```
uint MGLAPI MGL_getLineStippleCount(void)
```

Prototype In

mgraph.h

Description

Return the current line stipple counter. Refer to *MGL_setLineStipple* for more information on stippled lines.

See Also

MGL_setLineStyle, *MGL_setLineStipple*, *MGL_setLineStippleCount*

MGL_getLineStyle

Returns the current line style.

Declaration

```
int MGLAPI MGL_getLineStyle(void)
```

Prototype In

mgraph.h

Description

Returns the current line style. Refer to *MGL_setLineStipple* for more information on stippled lines.

See Also

MGL_setLineStyle, *MGL_setLineStipple*, *MGL_setLineStippleCount*

MGL_getPCXSize

Obtain the dimensions of a PCX file from disk.

Declaration

```
ibool MGLAPI MGL_getPCXSize(
    const char *PCXName,
    int *width,
    int *height,
    int *bitsPerPixel)
```

Prototype In

mgraph.h

Parameters

<i>PCXName</i>	Name of the bitmap file to load header for
<i>width</i>	Place to store the bitmap width
<i>height</i>	Place to store the bitmap height
<i>bitsPerPixel</i>	Place to store the bitmap pixel depth

Return Value

True if the PCX file was found, false if not.

Description

This functions loads all the header information for a PCX file from disk, without actually loading the bits for the bitmap surface. This is useful to determine the dimensions and pixel format for the bitmap before it is loaded, so you can create an appropriate memory device context that you can load the bitmap into with the *MGL_loadPCXIntoDC* function.

See Also

MGL_loadPCX, *MGL_loadPCXIntoDC*

MGL_getPCXSizeExt

Obtain the dimensions of a PCX file from an opened file.

Declaration

```
ibool MGLAPI MGL_getPCXSizeExt(
    FILE *f,
    ulong dwOffset,
    ulong dwSize,
    int *width,
    int *height,
    int *bitsPerPixel)
```

Prototype In

mgraph.h

Parameters

<i>f</i>	Open binary file to read bitmap from
<i>dwOffset</i>	Offset to start of PCX file
<i>dwSize</i>	Size of PCX file
<i>width</i>	Width of bitmap
<i>height</i>	Height of bitmap
<i>bitsPerPixel</i>	Pixel depth of bitmap

Return Value

True if PCX was found, false if not.

Description

This function is the same as *MGL_getPCXSize*, however it works with a previously opened file. This allows you to create your own large files with multiple files embedded in them.

See Also

MGL_getPCXSize

MGL_getPNGSize

Obtain the dimensions of a PNG file from disk.

Declaration

```
ibool MGLAPI MGL_getPNGSize(
    const char *PNGName,
    int *width,
    int *height,
    int *bitsPerPixel,
    pixel_format_t *pf)
```

Prototype In

mgraph.h

Parameters

<i>PNGName</i>	Name of the bitmap file to load header for
<i>width</i>	Place to store the bitmap width
<i>height</i>	Place to store the bitmap height
<i>bitsPerPixel</i>	Place to store the bitmap pixel depth
<i>pf</i>	Place to store the bitmap pixel format information

Return Value

True if the PNG file was found, false if not.

Description

This functions loads all the header information for a PNG file from disk, without actually loading the bits for the bitmap surface. This is useful to determine the dimensions and pixel format for the bitmap before it is loaded, so you can create an appropriate memory device context that you can load the bitmap into with the *MGL_loadPNGIntoDC* function.

See Also

MGL_loadPNG, *MGL_loadPNGIntoDC*

MGL_getPNGSizeExt

Obtain the dimensions of a PNG file from an opened file.

Declaration

```
ibool MGLAPI MGL_getPNGSizeExt(
    FILE *f,
    ulong dwOffset,
    ulong dwSize,
    int *width,
    int *height,
    int *bitsPerPixel,
    pixel_format_t *pf)
```

Prototype In

mgraph.h

Parameters

<i>f</i>	Open binary file to read bitmap from
<i>dwOffset</i>	Offset to start of PNG file
<i>dwSize</i>	Size of PNG file
<i>width</i>	Width of bitmap
<i>height</i>	Height of bitmap
<i>bitsPerPixel</i>	Pixel depth of bitmap
<i>pf</i>	Place to store the bitmap pixel format information

Return Value

True if PNG was found, false if not.

Description

This function is the same as *MGL_getPNGSize*, however it works with a previously opened file. This allows you to create your own large files with multiple files embedded in them.

See Also

MGL_getPNGSize

MGL_getPalette

Returns the currently active palette values.

Declaration

```
void MGLAPI MGL_getPalette(
    MGLDC *dc,
    palette_t *pal,
    int numColors,
    int startIndex)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context of interest
<i>pal</i>	Place to store the retrieved values
<i>numColors</i>	Number of color values to retrieve
<i>startIndex</i>	Starting palette index value to retrieve

Description

This function copies part or all of the currently active palette values and stores it in the array *pal*. You can specify only a subset of the palette values to be obtained with the *startIndex* and *numColors* arguments.

Thus to save the entire palette in a 256 color video mode, you would use (assuming enough space for the palette has been allocated):

```
MGL_getPalette(pal, 255, 0);
```

or to get the top half of the palette you would use:

```
MGL_getPalette(pal, 128, 128);
```

You should ensure that you have allocated enough memory to hold all of the palette values that you wish to read. You can use *MGL_getPaletteSize* to determine the size required to save the entire palette.

See Also

MGL_getPaletteEntry, *MGL_setPalette*, *MGL_getDefaultPalette*

MGL_getPaletteEntry

Returns the value of a single palette entry.

Declaration

```
void MGLAPI MGL_getPaletteEntry(
    MGLDC *dc,
    int entry,
    uchar *red,
    uchar *green,
    uchar *blue)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context of interest
<i>entry</i>	Palette index to read
<i>red</i>	Place to store the red component
<i>green</i>	Place to store the green component
<i>blue</i>	Place to store the blue component

Description

This function returns the value of a single color palette entry. If you wish to obtain more than a single palette entry you should use the *MGL_getPalette* routine which is faster for multiple entries.

See Also

MGL_setPaletteEntry, *MGL_getPalette*, *MGL_setPalette*

MGL_getPaletteSize

Returns the number of entries in the entire palette.

Declaration

```
int MGLAPI MGL_getPaletteSize(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context of interest

Return Value

Number of entries in entire palette.

Description

This function returns the number of entries in the entire palette. You should use this function to determine the size of the entire palette, since the palette is still available in HiColor and TrueColor video modes. For RGB modes the palette is implemented in software rather than hardware, and is used for translating color index values to RGB color values, such as when displaying color index bitmaps in RGB modes.

See Also

MGL_getPalette

MGL_getPaletteSnowLevel

Returns the current palette snow level.

Declaration

```
int MGLAPI MGL_getPaletteSnowLevel(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context of interest

Return Value

Current palette snow level.

Description

This function returns the number of palette entries that can be programmed during a single vertical retrace before the onset of snow. By default MGL programs use all 256 entries per retrace, but you may need to slow this down on systems with slower hardware that causes snow during multiple palette realization commands.

See Also

MGL_setPaletteSnowLevel

MGL_getPenBitmapPattern

Returns the currently active bitmap pattern index, and pattern data.

Declaration

```
int MGLAPI MGL_getPenBitmapPattern(
    int index,
    pattern_t *pat)
```

Prototype In

mgraph.h

Parameters

<i>index</i>	Index of the bitmap pattern slot to read from
<i>pat</i>	Place to store the bitmap pattern

Return Value

Currently active bitmap pattern index.

Description

This function returns a copy of the specified bitmap pattern used when rendering patterned primitives in the MGL_BITMAP_TRANSPARENT and MGL_BITMAP_OPQAUE pen styles. It also returns the currently active bitmap pattern index, since the MGL supports 8 patterns cached internally in the driver.

Note: *You can pass a value of NULL in the pat parameter, and the pattern data will not be copied. Ie: MGL_getPenBitmapPattern(0,NULL) returns just the currently active bitmap pattern index.*

See Also

MGL_setPenBitmapPattern, MGL_usePenBitmapPattern, MGL_setPenPixmapPattern, MGL_setPenStyle

MGL_getPenPixmapPattern

Returns the currently active pixmap pattern index, and pattern data.

Declaration

```
int MGLAPI MGL_getPenPixmapPattern(
    int index,
    pixmap_t *pat)
```

Prototype In

mgraph.h

Parameters

<i>index</i>	Index of the bitmap pattern slot to read from
<i>pat</i>	Place to store the pixmap pattern

Return Value

Currently active bitmap pattern index.

Description

This function returns a copy of the specified pixmap pattern used when rendering patterned primitives in the MGL_PIXMAP pen style. It also returns the currently active pixmap pattern index, since the MGL supports 8 patterns cached internally in the driver.

Note: *You can pass a value of NULL in the pat parameter, and the pattern data will not be copied. Ie: MGL_getPenPixmapPattern(0,NULL) returns just the currently active pixmap pattern index.*

See Also

MGL_setPenPixmapPattern, MGL_usePenPixmapPattern, MGL_setPenBitmapPattern, MGL_setPenStyle

MGL_getPenPixmapTransparent

Returns the currently active pixmap pattern transparent color.

Declaration

```
color_t MGLAPI MGL_getPenPixmapTransparent(void)
```

Prototype In

mgraph.h

Return Value

Returns the currently active pixmap pattern transparent color.

Description

This function returns the currently active pixmap pattern transparent color. This is used to determine which pixels in the pixmap pattern are transparent when the device context has the MGL_PIXMAP_TRANSPARENT pen style active.

See Also

MGL_setPenPixmapPattern, MGL_setPenPixmapTransparent

MGL_getPenSize

Returns the current pen size.

Declaration

```
void MGLAPI MGL_getPenSize(  
    int *height,  
    int *width)
```

Prototype In

mgraph.h

Parameters

<i>height</i>	Place to store the current pen height
<i>width</i>	Place to store the current pen width

Description

Return the size of the current pen in pixels. The default pen is 1 pixel by 1 pixel in dimensions, however you can change this to whatever value you like. When primitives are rasterized with a pen other than the default, the pixels in the pen always lie to the right and below the current pen position.

See Also

MGL_setPenSize

MGL_getPenStyle

Returns the current pen style.

Declaration

```
int MGLAPI MGL_getPenStyle(void)
```

Prototype In

mgraph.h

Return Value

Current pen style.

Description

This function returns the currently active pen style. Pen styles supported by the SciTech MGL are enumerated in *MGL_penStyleType*.

When filling in the MGL_BITMAP_ TRANSPARENT mode, the foreground color is used to fill in all pixels in the bitmap pattern that are a 1. Where the pixels in the bitmap pattern are a 0, the original background color is retained. In the MGL_BITMAP_ OPAQUE mode, the background color is used to fill in the pixels in the bitmap that are set to a 0. When filling in MGL_PIXMAP mode, the foreground and background color values are not used, and the pixel colors are obtained directly from the pixmap pattern colors.

See Also

MGL_setPenStyle, MGL_setPenBitmapPattern, MGL_setPenPixmapPattern

MGL_getPixel

Returns the color of a specified pixel.

Declaration

```
color_t MGL_getPixel(  
    point_t p)
```

Prototype In

mgraph.h

Parameters

p Coordinate of the pixel to read

Return Value

Color of the specified pixel.

Description

This function is the same as *MGL_getPixelCoord*, however it takes the coordinate of the pixel to read as a point.

See Also

MGL_getPixelCoord, *MGL_beginPixel*, *MGL_endPixel*

MGL_getPixelCoord

Returns the color of a specified pixel.

Declaration

```
color_t MGLAPI MGL_getPixelCoord(  
    int x,  
    int y)
```

Prototype In

mgraph.h

Parameters

x x coordinate of the pixel to read
y y coordinate of the pixel to read

Return Value

Color of the specified pixel.

Description

This function returns the color of the pixel at the specified coordinate.

See Also

MGL_getPixel, *MGL_beginPixel*, *MGL_endPixel*

MGL_getPixelCoordFast

Returns the color of a specified pixel.

Declaration

```
color_t MGLAPI MGL_getPixelCoordFast(  
    int x,  
    int y)
```

Prototype In

mgraph.h

Parameters

x x coordinate of the pixel to read
y y coordinate of the pixel to read

Return Value

Color of the specified pixel.

Description

This function returns the color of the pixel at the specified coordinate. Note that you must ensure that you call the routine *MGL_beginPixel* before reading any pixel values and the routine *MGL_endPixel* after reading a bunch of pixels with these fast pixel routines.

See Also

MGL_getPixelFast, *MGL_beginPixel*, *MGL_endPixel*

MGL_getPixelFast

Returns the color of a specified pixel.

Declaration

```
color_t MGL_getPixelFast(  
    point_t p)
```

Prototype In

mgraph.h

Parameters

p Coordinate of the pixel to read

Return Value

Color of the specified pixel.

Description

This function is the same as *MGL_getPixelCoordFast*, however it takes the coordinate of the pixel to read as a point.

See Also

MGL_getPixelCoordFast, *MGL_beginPixel*, *MGL_endPixel*

MGL_getPixelFormat

Returns the current packed pixel format information.

Declaration

```
void MGLAPI MGL_getPixelFormat(  
    MGLDC *dc,  
    pixel_format_t *pf)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context of interest
<i>pf</i>	Place to store the pixel format information

Description

This function returns the current pixel format information for the specified device context. This information is used by MGL to encode the packed pixel information, and can be used by your application to work out how to pack values correctly for the RGB device contexts.

See Also

MGL_packColor, *MGL_unpackColor*

MGL_getPlaneMask

Returns the current plane mask for all rendering operations.

Declaration

```
ulong MGLAPI MGL_getPlaneMask(void)
```

Prototype In

mgraph.h

Return Value

Current plane mask.

Description

This function returns the current plane mask for all rendering operations, which is used to mask out specific bits from being affected during writes to the framebuffer.

See Also

MGL_setPlaneMask

MGL_getPolygonType

Returns the current polygon type.

Declaration

```
int MGLAPI MGL_getPolygonType(void)
```

Prototype In

mgraph.h

Return Value

Current polygon type code.

Description

Returns the current polygon type. You can change this value with the *MGL_setPolygonType* to force MGL to work with a specific polygon type (and to avoid the default automatic polygon type checking). Polygon types supported by the SciTech MGL are enumerated in *MGL_polygonType*.

If you expect to be drawing lots of complex or convex polygons, setting the polygon type can result in faster polygon rasterizing. Note that this setting does not affect the specialized triangle and quadrilateral rasterizing routines.

See Also

MGL_setPolygonType, *MGL_fillPolygon*

MGL_getSpaceExtra

Returns the current space extra value.

Declaration

```
int MGLAPI MGL_getSpaceExtra(void)
```

Prototype In

mgraph.h

Return Value

Current space extra value.

Description

Returns the current space extra value used when drawing text in the current font. The space extra value is normally zero, but can be a positive or negative value. This value can be used to insert extra space between the characters in a font (making this value a large negative value will make the characters run on top of each other).

See Also

MGL_setSpaceExtra, *MGL_drawStr*

MGL_getTextDirection

Returns the current text direction.

Declaration

```
int MGLAPI MGL_getTextDirection(void)
```

Prototype In

mgraph.h

Return Value

Current text direction.

Description

Returns the current text direction. Directions supported by the SciTech MGL are enumerated in *MGL_textJustType*.

See Also

MGL_setTextDirection, *MGL_drawStr*

MGL_getTextJustify

Returns the current text justification.

Declaration

```
void MGLAPI MGL_getTextJustify(  
    int *horiz,  
    int *vert)
```

Prototype In

mgraph.h

Parameters

<i>horiz</i>	Place to store horizontal justification
<i>vert</i>	Place to store vertical justification

Description

Returns the current text justification values. Justification types supported by the SciTech MGL are enumerated in *MGL_textJustType*.

See Also

MGL_setTextJustify

MGL_getTextSettings

Declaration

```
void MGLAPI MGL_getTextSettings(  
    text_settings_t *settings)
```

Prototype In

mgraph.h

Parameters

settings Place to store the current text settings

Description

Returns a copy of the currently active text settings. This routine provides a way to save and restore all the values relating to the rasterizing of text in MGL with a single function call.

See Also

MGL_setTextSettings

MGL_getTextSize

Returns the current text scaling factors.

Declaration

```
void MGLAPI MGL_getTextSize(  
    int *numerx,  
    int *denomx,  
    int *numery,  
    int *denomy)
```

Prototype In

mgraph.h

Parameters

<i>numerx</i>	Place to store the x numerator value
<i>denomx</i>	Place to store the x denominator value
<i>numery</i>	Place to store the y numerator value
<i>denomy</i>	Place to store the y denominator value

Description

Returns the current text scaling factors used by MGL. The text size values define an integer scaling factor to be used, where the actual values will be computed using the following formula:

Note: *MGL can only scale vector fonts. Bitmap fonts cannot be scaled.*

See Also

MGL_setTextSize

MGL_getViewport

Returns the currently active viewport.

Declaration

```
void MGLAPI MGL_getViewport(  
    rect_t *view)
```

Prototype In

mgraph.h

Parameters

view Place to store the current viewport

Description

This function returns a copy of the currently active viewport. These dimensions are global to the entire device context surface. When the viewport is changed with this function, the viewport origin is reset to (0,0).

All output in MGL is relative to the current viewport, so by changing the viewport to a new value you can make all output appear in a different rectangular portion of the device surface.

See Also

MGL_getViewportDC, MGL_setViewport, MGL_setRelViewport, MGL_setViewportOrg, MGL_clearViewport, MGL_setClipRect

MGL_getViewportDC

Returns the currently active viewport for a specific DC.

Declaration

```
void MGLAPI MGL_getViewportDC(  
    MGLDC *dc,  
    rect_t *view)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to change viewport for
<i>view</i>	Place to store the current viewport

Description

This function is the same as *MGL_getViewport*, however the device context does not have to be the current device context.

See Also

MGL_getViewport, *MGL_setViewport*, *MGL_setRelViewport*, *MGL_setViewportOrg*,
MGL_clearViewport, *MGL_setClipRect*

MGL_getViewportOrg

Returns the current viewport origin.

Declaration

```
void MGLAPI MGL_getViewportOrg(  
    point_t *org)
```

Prototype In

mgraph.h

Parameters

org Place to store the viewport origin

Description

This function returns a copy of the currently active viewport origin. When a new viewport is set with the *MGL_setViewport* function, the viewport origin is reset to (0,0), which means that any primitives drawn at pixel location (0,0) will appear at the top left hand corner of the viewport.

You can change the logical coordinate of the viewport origin to any value you please, which will effectively offset all drawing within the currently active viewport. Hence if you set the viewport origin to (10,10), drawing a pixel at (10,10) would make it appear at the top left hand corner of the viewport.

See Also

MGL_getViewportOrgDC, *MGL_setViewport*, *MGL_setViewportOrg*

MGL_getViewportOrgDC

Returns the current viewport origin for a specific DC.

Declaration

```
void MGLAPI MGL_getViewportOrgDC(  
    MGLDC *dc,  
    point_t *org)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to change viewport for
<i>org</i>	Place to store the viewport origin

Description

This function is the same as *MGL_getViewportOrg*, however the device context does not have to be the current device context.

See Also

MGL_getViewportOrg, *MGL_setViewport*, *MGL_setViewportOrg*

MGL_getVisualPage

Returns the currently visible hardware video page.

Declaration

```
int MGLAPI MGL_getVisualPage(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context of interest

Return Value

Currently visible hardware video page.

Description

This function returns the currently visible hardware video page number for the given device context. The first hardware video page is number 0, the second is 1 and so on. The number of available hardware video pages depends on the type of underlying hardware, the video mode resolution and amount of video memory installed. Thus not all video modes support multiple hardware video pages.

See Also

MGL_setVisualPage, MGL_getActivePage, MGL_setActivePage

MGL_getWriteMode

Returns the current write mode operation.

Declaration

```
int MGLAPI MGL_getWriteMode(void)
```

Prototype In

mgraph.h

Return Value

Current write mode operation.

Description

Returns the currently active write mode. Write modes supported by the SciTech MGL for all output primitives are enumerated in *MGL_writeModeType*.

See Also

MGL_setWriteMode

MGL_getX

Returns the x coordinate of the current position.

Declaration

```
int MGLAPI MGL_getX(void)
```

Prototype In

mgraph.h

Return Value

x coordinate of current position

Description

Returns the x coordinate of the current position (CP). The CP is the current graphics cursor position, and is used by a number of output routines to determine where to begin drawing.

See Also

MGL_getY, MGL_getCP,

MGL_getY

Returns the y coordinate of the current position.

Declaration

```
int MGLAPI MGL_getY(void)
```

Prototype In

mgraph.h

Return Value

y coordinate of current position

Description

Returns the y coordinate of the current position (CP). The CP is the current graphics cursor position, and is used by a number of output routines to determine where to begin drawing.

See Also

MGL_getX, *MGL_getCP*

MGL_glChooseVisual

Choose an OpenGL visual to best match the passed in visual.

Declaration

```
void MGLAPI MGL_glChooseVisual(  
    MGLDC *dc,  
    MGLVisual *visual)
```

Parameters

<i>dc</i>	MGL device context
<i>visual</i>	Structure containing OpenGL visual information

Description

This function examines the visual passed in and modifies the values to best match the capabilities of the underlying OpenGL implementation. If a requested capability is not supported, the structure will be modified for the capabilities that the OpenGL implementation does support (ie: lowering the depth buffer size to 16 bits etc).

See Also

MGL_glSetVisual, MGL_glCreateContext

MGL_glCreateContext

Creates the OpenGL rendering context for the MGL device context.

Declaration

```
ibool MGLAPI MGL_glCreateContext(
    MGLDC *dc,
    int flags)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	MGL device context to enable OpenGL API for
<i>flags</i>	Flags to force type of OpenGL library used

Return Value

True if context was created successfully, false otherwise

Description

This function creates an OpenGL rendering context for the MGL device context, and enables support for OpenGL functions for the MGL device context. To provide a quick and easy way to get the MGL up and running with OpenGL support, you can pass in some simple flags that let the MGL know what OpenGL features you want enabled in the OpenGL visual for the MGL device context, by combining values in the MGL_glContextType enumeration. For instance if you wanted to start OpenGL with support for an RGB, double buffered and z-buffered mode you would pass in:

MGL_GL_RGB | MGL_GL_DOUBLE | MGL_GL_DEPTH

If you wish to have complete control over the OpenGL visual that is used for the MGL device context, you can call *MGL_glChooseVisual* and *MGL_glSetVisual* before calling *MGL_glCreateContext*, and then pass in a value of MGL_GL_VISUAL to tell the MGL to use the visual you have already set for the device context instead of trying to create one from the passed in flags.

Note: *Once you have created an OpenGL rendering context for the MGL device context, you must first call MGL_glMakeCurrent to make that specific MGL device context the current OpenGL rendering context before you call any core OpenGL API functions.*

Note: *You must call this function first before you attempt to make any calls to the core OpenGL API functions (such as glVertex3f etc), as the MGL will not have initialised its internal dispatch tables until this call is made.*

See Also

MGL_glChooseVisual, MGL_glSetVisual, MGL_glMakeCurrent

MGL_glDeleteContext

Delete the OpenGL rendering context associated with the MGL device contex.

Declaration

```
void MGLAPI MGL_glDeleteContext(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc MGL device context to delete OpenGL rendering context for

Description

This function destroys the OpenGL rendering context for the MGL device context, and calls OpenGL to ensure that no rendering context is still current. You must call this function before you destroy and MGL device context if you have enabled OpenGL via *MGL_glCreateContext*.

Note: *After you have called this function, it is an error to make calls to the OpenGL API as OpenGL will not have a current rendering context active.*

See Also

MGL_glCreateContext, MGL_glMakeCurrent

MGL_glDisableMGLFuncs

Disables MGL 2D drawing functions using an OpenGL surface

Declaration

```
void MGLAPI MGL_glDisableMGLFuncs (MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context to disable 2D MGL functions for

Description

This function disable support for MGL 2D functions for hardware accelerated OpenGL surfaces. This does the opposite of the *MGL_glEnableMGLFuncs* function.

See Also

MGL_glEnableMGLFuncs

MGL_glEnableMGLFuncs

Enables MGL 2D drawing functions using an OpenGL surface

Declaration

```
void MGLAPI MGL_glEnableMGLFuncs (
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context to enable 2D MGL functions for

Description

This function enable support for MGL 2D functions for hardware accelerated OpenGL surfaces. In order to be able to draw to a hardware OpenGL surface using 2D OpenGL drawing functions, the MGL needs to re-program the state of the OpenGL rendering context such that it maps to a 2D integer coordinate system rather than the 2D or 3D coordinate system the user application code is using for OpenGL drawing. Hence this function saves the the state of the OpenGL rendering context so it can be restored with *MGL_glDisableMGLFuncs*, and sets the rendring context into a state that is suitable for 2D drawing via internal functions in the MGL.

Although it is not necessary to call this function to use the MGL 2D drawing functions (such as *MGL_fillRect*, *MGL_bitBlt*, *MGL_drawStr* etc), the process of saving and restoring the rendering context is expensive. Internally the MGL will call this function before doing any 2D operations, which means the state is saved and restore for every 2D MGL function that is called. Hence if you will be calling a number of 2D MGL drawing functions, you can bracket your code with *MGL_glEnableMGLFuncs* and *MGL_glDisableMGLFuncs* to ensure that the MGL only saves and restores the rendering context once for all MGL 2D drawing functions you call.

See Also

MGL_glDisableMGLFuncs

MGL_glEnumerateDrivers

Enumerates the names of all available OpenGL implementations.

Declaration

```
const char ** MGLAPI MGL_glEnumerateDrivers(void)
```

Prototype In

mgraph.h

Return Value

Pointer to a NULL terminated list of available OpenGL driver names.

Description

This function returns a NULL terminated list of OpenGL driver names to the application program, which describes all the available OpenGL drivers on the system. Once you have obtained the driver names, you can change the current OpenGL driver with *MGL_glSetDriver*.

Note that this function will always returns built in names for 'auto', 'microsoft', 'sgi' and 'mesa' and setting a driver to these names corresponds to changing the OpenGL driver type with *MGL_glSetOpenGLType*. However if there are multiple fullscreen hardware OpenGL drivers on the system and registered with the MGL, they will be listed one after the other. For example if you have an ATI 3DRage, a 3Dfx Voodoo and an NEC PowerVR installed in your system and there are fullscreen OpenGL drivers for all these boards registered with the MGL, the names for these drivers would be returned in this list also.

See Also

MGL_glSetDriver, *MGL_glSetOpenGLType*

MGL_glGetProcAddress

Returns the address of an OpenGL extension function.

Declaration

```
void * MGLAPI MGL_glGetProcAddress (
    const char *procName)
```

Prototype In

mgraph.h

Return Value

Address of the specified extension function, NULL if not available.

Description

This function returns the address of an OpenGL extension function if that extension is supported by the OpenGL implementation. Each OpenGL implementation may export a number of OpenGL extension that may not be supported by other OpenGL implementations, and this function is the mechanism you can use to obtain the address of those extension functions.

Note that you should first check to see if an extension is available, but calling the OpenGL function `glGetString(GL_EXTENSIONS)` to get a list of all the available extensions. In order to check for a specific extension by name, you can use the following code:

```
ibool checkExtension(const char *name)
{
    const char *p = (const char *)glGetString(GL_EXTENSIONS);
    while (p = strstr(p, name)) {
        const char *q = p + strlen(name);
        if (*q == ' ' || *q == '\\0')
            return GL_TRUE;
        p = q;
    }
    return GL_FALSE;
}
```

Note: *It is an error to call this function for an MGL device context that does not have an OpenGL rendering context associated with it via a call to `MGL_glCreateContext`.*

MGL_glGetVisual

Returns the current OpenGL visual for the device context.

Declaration

```
void MGLAPI MGL_glGetVisual(  
    MGLDC *dc,  
    MGLVisual *visual)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	MGL device context
<i>visual</i>	Place to store the OpenGL visual information

Description

This function returns the current OpenGL visual that has been set for the MGL device context.

See Also

MGL_glSetVisual, *MGL_glCreateContext*

MGL_glHaveHWOpenGL

Checks for OpenGL hardware acceleration.

Declaration

```
ibool MGLAPI MGL_glHaveHWOpenGL(void)
```

Prototype In

mgraph.h

Return Value

True if OpenGL is hardware accelerated on the target platform.

Description

This function will load the system OpenGL libraries and attempt to determine if hardware acceleration is available or not. On the Win32 platform, this involves loading the Microsoft OpenGL libraries and checking to see if OpenGL is accelerated via an ICD or MCD hardware device driver. On other platforms we check to see if acceleration is available via the system OpenGL drivers. This function is most useful for determining if OpenGL is hardware accelerated when running in windowed modes.

MGL_glMakeCurrent

Makes a MGL device context the current OpenGL rendering context.

Declaration

```
void MGLAPI MGL_glMakeCurrent(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context to make the current OpenGL rendering context.

Description

This function makes the MGL device context the current rendering context for all OpenGL rendering commands. You must first call *MGL_glCreateContext* for the MGL device context to create a valid OpenGL rendering context before you can call this function to make it the rendering current OpenGL rendering context.

The MGL and OpenGL allow you to create multiple rendering context, and to switch between them for output you must use this function to make one of them current at a time. You cannot have more than one OpenGL rendering context current at one time. For instance you may be drawing to a fullscreen OpenGL rendering context, but have an MGL memory device context that you wish to render a small 3D scene into, so you would make the memory device context the current OpenGL rendering context with a call to this function. The any subsequent OpenGL commands will draw to the memory device context instead of the display device context.

See Also

MGL_glCreateContext, *MGL_glDeleteContext*

MGL_glRealizePalette

Realizes the hardware palette for a device context when using OpenGL.

Declaration

```
void      MGLAPI MGL_glRealizePalette(MGLDC *dc,int numColors,int
startIndex,ibool waitVRT);
void MGLAPI MGL_glRealizePalette(
    MGLDC *dc,
    int numColors,
    int startIndex,
    int waitVRT)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to set palette for
<i>numColors</i>	Number of colors in the palette
<i>startIndex</i>	Starting index in the palette
<i>waitVRT</i>	True if routine should sync to vertical retrace, false if not.

Description

This function realizes the hardware palette associated with a display device context. Calls to *MGL_glSetPalette* only update the palette values in the color palette for the device context structure, but do not actually program the hardware palette for display device contexts in 4 and 8 bits per pixel modes. In order to program the hardware palette you must call this routine.

When the hardware palette is realized, you normally need to sync to the vertical retrace to ensure that the palette values are programmed without the onset of snow (see *MGL_setPaletteSnowLevel* to adjust the number of colors programmed per retrace period). If however you wish to perform double buffered animation and change the hardware color palette at the same time, you should call this routine immediately after calling either *MGL_setVisualPage* or *MGL_swapBuffers* with the waitVRT flag set to false.

Note: *This function is identical to the regular MGL_realizePalette function, however if you are running OpenGL you must use this function instead.*

See Also

MGL_glSetPalette, *MGL_realizePalette*

MGL_glResizeBuffers

Resizes the OpenGL buffers for the windowed device context.

Declaration

```
void MGLAPI MGL_glResizeBuffers(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc MGL device context to resize the buffers for

Description

This function informs the MGL that the dimensions of a windowed device context have changed, and that the OpenGL rendering buffers need to be re-sized to the new dimensions of the window. It is up to the application programmer to capture the WM_SIZE messages in windowed modes, and call this function when the window size changes to let the MGL correctly update the buffer dimensions.

Note: *This function is not necessary in fullscreen modes.*

MGL_glSetDriver

Sets the currently active OpenGL driver

Declaration

```
ibool MGLAPI MGL_glSetDriver(  
    const char *name)
```

Prototype In

mgraph.h

Parameters

name Name of the OpenGL driver to make the current driver

Return Value

True on success, false on failure.

Description

This function changes the currently active OpenGL driver to the name you pass in. Note that the name you pass in must be one of the names returned by the *MGL_glEnumerateDrivers*, or the function will fail.

See Also

MGL_glEnumerateDrivers, *MGL_glSetOpenGLType*

MGL_glSetOpenGLType

Sets the OpenGL implementation type to use.

Declaration

```
void MGLAPI MGL_glSetOpenGLType(  
    int type)
```

Prototype In

mgraph.h

Parameters

type New OpenGL type to set (*MGL_glOpenGLType*)

Description

This function sets the current OpenGL implementation type to use according to the passed in type parameter (*MGL_glOpenGLType* enumeration). In the AUTO mode we automatically determine which version of OpenGL to use depending on the target runtime system. Unless there is hardware acceleration available we choose Silicon Graphic's OpenGL, but if hardware acceleration is present we use the system provided OpenGL implementation so we can utilize the hardware (which is Microsoft OpenGL for Win32).

Note: *If you wish to be able to choose between the registered MGL hardware OpenGL drivers within your application, use the MGL_glEnumerateDrivers and MGL_glSetDriver functions instead.*

See Also

MGL_glSetDriver, MGL_glEnumerateDrivers

MGL_glSetPalette

Sets the palette values for a device context when using OpenGL.

Declaration

```
void MGLAPI MGL_glSetPalette(
    MGLDC *dc,
    palette_t *pal,
    int numColors,
    int startIndex)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to set palette for
<i>pal</i>	Palette to set
<i>numColors</i>	Number of colors in the palette
<i>startIndex</i>	Starting index in the palette

Description

This functions sets the color palette for an MGL device context when running in OpenGL rendering modes. This function is identical to the regular *MGL_setPalette* function, however if you are running OpenGL you must use this function instead.

See Also

MGL_glRealizePalette, *MGL_setPalette*

MGL_glSetVisual

Attempts to set the specified OpenGL visual for the MGL device context.

Declaration

```
ibool MGLAPI MGL_glSetVisual (
    MGLDC *dc,
    MGLVisual *visual)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	MGL device context
<i>visual</i>	Structure containing OpenGL visual information

Return Value

True on success, false if visual not supported by OpenGL implementation.

Description

This function sets the passed in OpenGL visual for the MGL device context and makes it the visual that will be used in the call to *MGL_glCreateContext*. Note that this function may fail if the OpenGL visual requested is invalid, and you should call *MGL_glChooseVisual* first to find a visual that best matches the underlying OpenGL implementation. For instance if the OpenGL implementation only supports a 16-bit z-buffer, yet you request a 32-bit z-buffer this function will fail.

The OpenGL visual is used to define the visual capabilities of the OpenGL rendering context that will be created with the *MGL_glCreateContext* function, and includes information such as whether the mode should be an RGB mode or color index mode, whether it should be single buffered or double buffered, whether a depth buffer (zbuffer) should be used and how many bits it should be etc.

Note: *You can only set the visual for a context once, and it is an error to call MGL_glSetVisual more than once for an MGL device context, and you also cannot change a visual once you have set it without first destroying the OpenGL rendering context.*

See Also

MGL_glChooseVisual, *MGL_glCreateContext*

MGL_glSwapBuffers

Swaps the display buffers for OpenGL rendering.

Declaration

```
void MGLAPI MGL_glSwapBuffers(  
    MGLDC *dc,  
    int waitVRT)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	MGL device context to swap display buffers for
<i>waitVRT</i>	True to wait for vertical retrace

Description

This function swaps the OpenGL rendering buffers, and the current back buffer becomes the front buffer and vice versa. If you are running in a window, the context of the back buffer will be copied to the current window. If you are running in a fullscreen graphics mode with hardware page flipping, this function will do a hardware page flip to swap the buffers.

When the OpenGL buffers are swapped, you should normally allow MGL/OpenGL driver to sync to the vertical retrace to ensure that the change occurs in the correct place, and that you don't get flicker effects on the display. You may however turn off the vertical retrace synching if you are synching up with the retrace period using some other means, or if you are measuring the performance of your application.

MGL_globalToLocal

Converts a point from global to local coordinates.

Declaration

```
void MGLAPI MGL_globalToLocal(  
    point_t *p)
```

Prototype In

mgraph.h

Parameters

p Pointer to point to be converted

Description

This function converts a coordinate from global coordinates to local coordinates. Global coordinates are defined relative to the entire output device context surface, while local coordinates are relative to the currently active viewport.

This routine is usually used to convert mouse coordinate values from global screen coordinates to the local coordinate system of the currently active viewport.

See Also

MGL_globalToLocalDC, *MGL_localToGlobal*

MGL_globalToLocalDC

Converts a point from global to local coordinates.

Declaration

```
void MGLAPI MGL_globalToLocalDC(  
    MGLDC *dc,  
    point_t *p)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context in which the point is defined
<i>p</i>	Pointer to point to be converted

Description

This function is the same as *MGL_globalToLocal*, however the device context does not have to be the current device context.

See Also

MGL_globalToLocal, *MGL_localToGlobal*

MGL_halfTonePixel

Compute color for pixel in halftone palette.

Declaration

```
uchar MGLAPI MGL_halfTonePixel(  
    int x,  
    int y,  
    uchar R,  
    uchar G,  
    uchar B)
```

Prototype In

mgraph.h

Parameters

<i>x</i>	x pixel coordinate
<i>y</i>	y pixel coordinate
<i>R</i>	Red component for pixel color
<i>G</i>	Green component for pixel color
<i>B</i>	Blue component for pixel color

Return Value

Color index for the pixel in MGL halftone palette

Description

This function computes the color index for a specified 24 bit RGB pixel value in the standard MGL halftone palette, and can be used to perform real-time dithering of pixel values from RGB colors to color index colors. This routine uses a fast table lookup and an 8x8 ordered dither to do the conversion.

See Also

MGL_getHalfTonePalette

MGL_halfTonePixel555

Compute color for an 555 format RGB pixel using a halftone dither

Declaration

```
ushort MGLAPI MGL_halfTonePixel555(
    int x,
    int y,
    uchar R,
    uchar G,
    uchar B)
```

Prototype In

mgraph.h

Parameters

<i>x</i>	x coordinate of pixel (need only be relative)
<i>y</i>	y coordinate of pixel (need only be relative)
<i>R</i>	R value for pixel (8 bit components)
<i>G</i>	G value for pixel (8 bit components)
<i>B</i>	B value for pixel (8 bit components)

Return Value

Packed RGB 555 format pixel color

Description

This function computes the color of an RGB 555 format pixel using a fast halftone dither algorithm. The resulting color is pre-packed into the correct framebuffer format as part of the dithering process.

See Also

MGL_halfTonePixel565

MGL_halfTonePixel565

Compute color for an 565 format RGB pixel using a halftone dither

Declaration

```
ushort MGLAPI MGL_halfTonePixel565(  
    int x,  
    int y,  
    uchar R,  
    uchar G,  
    uchar B)
```

Prototype In

mgraph.h

Parameters

<i>x</i>	x coordinate of pixel (need only be relative)
<i>y</i>	y coordinate of pixel (need only be relative)
<i>R</i>	R value for pixel (8 bit components)
<i>G</i>	G value for pixel (8 bit components)
<i>B</i>	B value for pixel (8 bit components)

Return Value

Packed RGB 565 format pixel color

Description

This function computes the color of an RGB 565 format pixel using a fast halftone dither algorithm. The resulting color is pre-packed into the correct framebuffer format as part of the dithering process.

See Also

MGL_halfTonePixel555

MGL_haveWidePalette

Tests whether the palette for a device context uses an 8-bit wide palette.

Declaration

```
ibool MGLAPI MGL_haveWidePalette(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Display device context of interest.

Return Value

True if palette is 8 bits wide, false if 6 bits wide.

Description

This function will return true if the display device context is using an 8-bit per primary hardware palette, or false if it is using a 6-bit per primary hardware palette. The original VGA standard only supports 6-bits per primary for the color palette, giving you the selection of 256 colors out of 256,000 for 8bpp modes. However more modern controllers provide support for 8-bits per primary, giving you a selection of 256 colors out of a total 16.7 million for 8bpp modes.

MGL_init

Initializes MGL graphics library.

Declaration

```
int MGLAPI MGL_init(
    const char *mglpath,
    const char *server)
```

Prototype In

mgraph.h

Parameters

<i>mglpath</i>	Path to standard MGL resource files
<i>server</i>	Name of server to connect to (NULL for local system)

Return Value

Number of display devices attached, or 0 on error.

Description

This function initializes MGL for use, and should always be the first function called in MGL applications. The *mglpath* parameter is used by the MGL to locate all MGL resource files in their standard locations. The value passed in here should point to the base directory where all the resources are located. This may be the current directory if all resources are relative to the current directory of the application (i.e. a value of "." passed in the *mglpath* parameter).

After you have called *MGL_init*, the MGL will automatically detect the installed graphics hardware when any of *MGL_availablePages*, *MGL_modeResolution*, *MGL_modeFlags* or *MGL_findMode* are called. The priority ordering of which driver will be used depends on the capabilities of the underlying display hardware, and the drivers will be chosen in the following order (first one in table is selected in preference to ones below it):

<i>Driver</i>	Highest performance driver selected
<i>Snap</i>	SciTech SNAP Graphics driver
<i>DirectDraw</i>	SciTech SNAP Graphics DirectDraw driver
<i>VBE2</i>	SciTech SNAP Graphics VBE/Core 2.0 driver
<i>VBE1</i>	SciTech SNAP Graphics VBE/Core 1.2 driver
<i>StandardVGA</i>	SciTech SNAP Graphics VGA driver

For instance if you had DirectDraw installed on your system and MGL found a SciTech SNAP Graphics driver, those modes supported by the SciTech SNAP Graphics drivers will use the SciTech SNAP Graphics driver in preference to the DirectDraw driver. If however the DirectDraw driver has additional modes not supported by the SciTech SNAP Graphics driver, the DirectDraw drivers would be used for those modes. If however the user has DirectDraw installed and only has a VBE 2.0 driver available, the DirectDraw drivers will be used for all modes whenever possible.

To change this default behavior you can selectively disable those drivers that you do not wish to be used immediately after calling *MGL_init*. A typical sequence of code to register drivers and allows the program to disable SciTech SNAP Graphics might be as follows:

```
int      mode;
MGLDC    *dc;

if (MGL_init(".",NULL) == 0)
    MGL_fatalError(MGL_errorMsg(MGL_result()));
MGL_disableDriver(MGL_SNAPNAME);
if ((mode = MGL_findMode(640,480,8)) == -1)
    MGL_fatalError(MGL_errorMsg(MGL_result()));
dc = MGL_createDisplayDC(mode,1,MGL_DEFAULT_REFRESH);
...
```

When the MGL is searching for resource files (bitmaps, icons, fonts and cursors), it will first attempt to find the files just by using the filename itself. Hence if you wish to look for a specific bitmap or font file, you should pass the full pathname to the file that you want. If the filename is a simple relative filename (i.e. "myfont.fnt"), MGL will then search in the standard directories relative to the path specified in *mglpath*. The standard locations that MGL will look for the resource files are as follows:

Resource	Base pathname
Bitmaps	<i>mglpath</i> /bitmaps
Fonts	<i>mglpath</i> /fonts
Icons	<i>mglpath</i> /icons
Cursors	<i>mglpath</i> /cursors

As a final resort the MGL will also look for the files relative to the *MGL_ROOT* environment variable. The *MGL_ROOT* variable can be set to point to a standard base directory where all MGL resources will be stored on the end user's machine. For MGL developer machines, *MGL_ROOT* is usually set to point to the directory where the SciTech MGL files were installed (ie: c:\scitech by default for DOS, Windows and OS/2 installations, \$HOME/scitech for Unix installations).

The server parameter is used to connect to a remote server to display the output of the of the MGL program on a remote machine. For remote connections, the server name should either be the DNS name or TCP/IP address of the machine to connect to. For local operation on the client machine, this parameter should simply be set to NULL.

Once you have initialised the MGL, you then need to find an appropriate display mode to use with *MGL_findDisplayMode*, and then create a device context for all output, using *MGL_createDisplayDC* (for fullscreen display device contexts). To support multiple display controllers, please see the documentation for *MGL_selectDisplayDevice* to initialise all installed display devices.

If anything goes wrong during the initialization process, MGL will return a result code via the *MGL_result* routine. You can then use this result code to determine the cause of

the problem, and use the `MGL_errorMsg` routine to display an appropriate error message for the user.

Note: *To get up and running quickly with your first MGL program, check out the `MGL_quickInit()` function which allows you to start the MGL with a single line of code! Or alternatively check out the Game Framework API if you are developing game type applications.*

Note: *Remote client/server operation is not supported at this time. This is something scheduled for a future release of the MGL.*

See Also

`MGL_quickInit`, `MGL_selectDisplayDevice`, `MGL_enableAllDrivers`, `MGL_disableDriver`, `MGL_availablePages`, `MGL_modeResolution`, `MGL_findMode`, `MGL_addCustomMode`, `MGL_createDisplayDC`, `MGL_exit`, `MGL_result`

MGL_insetRect

Insets the coordinates of a rectangle.

Declaration

```
void MGL_insetRect(  
    rect_t r,  
    int dx,  
    int dy)
```

Prototype In

mgraph.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

MGL_offsetRect

MGL_isCurrentDC

Determines if the specified device context is the currently active context.

Declaration

```
ibool MGLAPI MGL_isCurrentDC(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context to check

Return Value

True if DC is the currently active context, false otherwise.

Description

This function determines if the passed in device context is the currently active device context. The currently active device context is where all the output from all the MGL rendering functions will be displayed.

See Also

MGL_makeCurrentDC

MGL_isDisplayDC

Determines if the specified device context is a display device context.

Declaration

```
ibool MGLAPI MGL_isDisplayDC(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context to check

Return Value

True if the device context is a display DC, false if not.

Description

This function determines if the passed in device context is a fullscreen display device context, or some other type of device context.

See Also

MGL_isOffscreenDC, MGL_isOverlayDC, MGL_isMemoryDC, MGL_isWindowedDC

MGL_isMemoryDC

Determines if the specified device context is a memory device context.

Declaration

```
ibool MGLAPI MGL_isMemoryDC(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context to check

Return Value

True if the device context is a memory DC, false if not.

Description

This function determines if the passed in device context is a memory device context, or some other type of device context.

See Also

MGL_isDisplayDC, MGL_isOffscreenDC, MGL_isOverlayDC, MGL_isWindowedDC

MGL_isOffscreenDC

Determines if the specified device context is an offscreen device context.

Declaration

```
ibool MGLAPI MGL_isOffscreenDC(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context to check

Return Value

True if the device context is an offscreen DC, false if not.

Description

This function determines if the passed in device context is an offscreen device context, or some other type of device context.

See Also

MGL_isDisplayDC, MGL_isOverlayDC, MGL_isMemoryDC, MGL_isWindowedDC

MGL_isOverlayDC

Determines if the specified device context is an overlay device context.

Declaration

```
ibool MGLAPI MGL_isOverlayDC(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context to check

Return Value

True if the device context is an overlay DC, false if not.

Description

This function determines if the passed in device context is an overlay device context, or some other type of device context.

See Also

MGL_isDisplayDC, MGL_isOffscreenDC, MGL_isMemoryDC, MGL_isWindowedDC

MGL_isSimpleRegion

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

Declaration

```
ibool MGL_isSimpleRegion(  
    region_t r)
```

Prototype In

mgraph.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

MGL_unionRegion, MGL_diffRegion, MGL_sectRegion

MGL_isStereoDC

Determines if the specified device context is stereo enabled.

Declaration

```
ibool MGLAPI MGL_isStereoDC(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context to check

Return Value

True if the device context is stereo enabled, false if not.

Description

This function determines if the passed in device context supports stereo rendering.

See Also

MGL_isDisplayDC, MGL_isOffscreenDC, MGL_isOverlayDC, MGL_isWindowedDC

MGL_isVSync

Returns true if the vertical sync is currently active

Declaration

```
int MGLAPI MGL_isVSync(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Display device context of interest

Return Value

True if in vSync, false if not and -1 if function not available on hardware device.

Description

This function will wait until the next vertical sync comes along for the current fullscreen graphics mode, before returning.

Note: *This function may not be available on all display devices, so make sure you check the return value to see if the function is implemented and available.*

See Also

MGL_vSync, MGL_getCurrentScanLine

MGL_isWindowedDC

Determines if the specified device context is a display device context.

Declaration

```
ibool MGLAPI MGL_isWindowedDC(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context to check

Return Value

True if the device context is a windowed DC, false if not.

Description

This function determines if the passed in device context is a windowed device context, or some other type of device context.

See Also

MGL_isDisplayDC, MGL_isOffscreenDC, MGL_isOverlayDC, MGL_isMemoryDC

MGL_leftTop

Returns the left top point of a rectangle.

Declaration

```
point_t MGL_leftTop(  
    rect_t r)
```

Prototype In

mgraph.h

Parameters

r Rectangle to return left top corner coordinate for.

Return Value

Returns the top left coordinate of the rectangle.

See Also

MGL_rightBottom

MGL_line

Draws a line with integer coordinates.

Declaration

```
void MGL_line(  
    point_t p1,  
    point_t p2)
```

Prototype In

mgraph.h

Parameters

<i>p1</i>	First endpoint of line
<i>p2</i>	Second endpoint of line

Description

This function is the same as *MGL_lineCoord*, however it takes the coordinates of the line as two points.

See Also

MGL_lineFX, *MGL_lineCoord*, *MGL_lineCoordFX*

MGL_lineCoord

Draws a line with integer coordinates.

Declaration

```
void MGLAPI MGL_lineCoord(  
    int x1,  
    int y1,  
    int x2,  
    int y2)
```

Prototype In

mgraph.h

Parameters

<i>x1</i>	x coordinate for first endpoint
<i>y1</i>	y coordinate for first endpoint
<i>x2</i>	x coordinate for second endpoint
<i>y2</i>	y coordinate for second endpoint

Description

Draws a line starting at the point (x1,y1) and ending at the point (x2,y2) in the current pen style, color and dimensions. The CP is not updated, and the line is clipped to the current clipping rectangle if clipping is on. Note that this function takes the coordinates of the lines in integer format.

See Also

MGL_line, *MGL_lineCoordExt*

MGL_lineCoordExt

Draws a line with integer coordinates.

Declaration

```
void MGLAPI MGL_lineCoordExt (
    int x1,
    int y1,
    int x2,
    int y2,
    ibool drawLast)
```

Prototype In

mgraph.h

Parameters

<i>x1</i>	x coordinate for first endpoint
<i>y1</i>	y coordinate for first endpoint
<i>x2</i>	x coordinate for second endpoint
<i>y2</i>	y coordinate for second endpoint
<i>drawLast</i>	True to draw the last pixel in the line

Description

Draws a line starting at the point (x1,y1) and ending at the point (x2,y2) in the current pen style, color and dimensions. The CP is not updated, and the line is clipped to the current clipping rectangle if clipping is on. Note that this function takes the coordinates of the lines in integer format.

If the drawLast parameter is true, the last pixel in the line is drawn. If not the last pixel is skipped. This is useful for drawing polylines so that the lines join up correctly and do not cause over draw when drawing in non-replace mode.

See Also

MGL_line, *MGL_lineCoord*

MGL_lineEngine

Generates the set of integer points on a line, given integer coordinates.

Declaration

```
void MGLAPI MGL_lineEngine(
    int x1,
    int y1,
    int x2,
    int y2,
    void (MGLAPIP plotPoint) (
        long x,
        long y))
```

Prototype In

mgraph.h

Parameters

<i>x1</i>	X coordinate for first endpoint
<i>y1</i>	Y coordinate for first endpoint
<i>x2</i>	X coordinate for second endpoint
<i>y2</i>	Y coordinate for second endpoint
<i>plotPoint</i>	User supplied pixel plotting routine

Description

This function generates the set of points on a line, and calls a user supplied plotPoint routine for every point generated.

See Also

MGL_lineEngineFX, MGL_ellipseEngine, MGL_ellipseArcEngine

MGL_lineExt

Draws a line with integer coordinates.

Declaration

```
void MGL_lineExt(  
    point_t p1,  
    point_t p2,  
    ibool drawLast)
```

Prototype In

mgraph.h

Parameters

<i>p1</i>	First endpoint of line
<i>p2</i>	Second endpoint of line
<i>drawLast</i>	True to draw the last pixel in the line

Description

This function is the same as *MGL_lineCoordExt*, however it takes the coordinates of the line as two points.

See Also

MGL_lineFX, *MGL_lineCoordExt*, *MGL_lineCoord*, *MGL_lineCoordFX*

MGL_lineRel

Draws a relative line.

Declaration

```
void MGL_lineRel(  
    point_t p)
```

Prototype In

mgraph.h

Parameters

p Amount to offset in (x,y) coordinates

Description

This function is the same as *MGL_lineRelCoord*, however the amount to move the CP by is passed as a point.

See Also

MGL_line, *MGL_lineCoord*, *MGL_lineTo*, *MGL_lineToCoord*, *MGL_lineRelCoord*,
MGL_moveRel, *MGL_moveRelCoord*

MGL_lineRelCoord

Draws a relative line.

Declaration

```
void MGLAPI MGL_lineRelCoord(
    int dx,
    int dy)
```

Prototype In

mgraph.h

Parameters

dx Amount to offset in x coordinate
dy Amount to offset in y coordinate

Description

Draws a line from the current position (CP) to the relative location that is a distance of (dx,dy) away from the CP. Thus the location of the next point on the line is:

(CP.x + dx, CP.y + dy)

The CP is updated to this value.

See Also

MGL_line, *MGL_lineCoord*, *MGL_lineTo*, *MGL_lineToCoord*, *MGL_lineRel*, *MGL_moveRel*, *MGL_moveRelCoord*

MGL_lineTo

Draws a line from the CP to the specified point.

Declaration

```
void MGL_lineTo(  
    point_t p)
```

Prototype In

mgraph.h

Parameters

p Point to draw to

Description

This function is the same as *MGL_lineToCoord*, however the point to draw to is passed as a point.

See Also

MGL_line, *MGL_lineCoord*, *MGL_lineToCoord*, *MGL_lineRel*, *MGL_lineRelCoord*,
MGL_moveRel, *MGL_moveRelCoord*

MGL_lineToCoord

Draws a line from the CP to the specified point.

Declaration

```
void MGLAPI MGL_lineToCoord(  
    int x,  
    int y)
```

Prototype In

mgraph.h

Parameters

<i>x</i>	x coordinate to draw to
<i>y</i>	y coordinate to draw to

Description

Draws a line from the current position (CP) to the new point (x,y). The CP is set to the point (x,y) on return from this routine.

See Also

MGL_line, *MGL_lineCoord*, *MGL_lineTo*, *MGL_lineRel*, *MGL_lineRelCoord*, *MGL_moveRel*, *MGL_moveRelCoord*

MGL_loadBitmap

Load a lightweight bitmap file from disk.

Declaration

```
bitmap_t * MGLAPI MGL_loadBitmap(
    const char *bitmapName,
    ibool loadPalette)
```

Prototype In

mgraph.h

Parameters

<i>bitmapName</i>	Name of bitmap file to load
<i>loadPalette</i>	True if the palette should also be loaded

Return Value

Pointer to the loaded bitmap file, NULL on error.

Description

Locates the specified bitmap file and loads it into a lightweight bitmap structure. MGL can load any Windows 3.x style bitmap files, including new format bitmap files with colors depths of 15/16 and 32 bits per pixel. Consult the Windows SDK documentation for the format of Windows bitmap files.

If loadPalette is true, the palette values for the bitmap will be loaded into the structure as well (if there is no palette, it will not be loaded), otherwise the palette entry for the bitmap will be NULL. For small bitmaps you can save space by not loading the palette for the bitmap.

When MGL is searching for bitmap files it will first attempt to find the files just by using the filename itself. Hence if you wish to look for a specific bitmap file, you should pass the full pathname to the file that you are interested in. If the filename is a simple relative filename (i.e. "MYBMP.BMP"), MGL will then search in the BITMAPS directory relative to the path specified in mglpath variable that was passed to *MGL_init*. As a final resort MGL will also look for the files in the BITMAPS directory relative to the MGL_ROOT environment variable.

If the bitmap file was not found, or an error occurred while reading the bitmap file, this function will return NULL. You can check the *MGL_result* error code to determine the cause.

The routine allocates a lightweight bitmap structure for holding the bitmap, which loads the bitmap with the minimum memory overheads. You can draw the bitmap on any device context surface using the *MGL_putBitmap* function, but you don't have the full flexibility of using a full memory device context for the bitmap surface. If you need more control over the bitmap, you can allocate a memory device context to hold the bitmap surface and load the bitmap with the *MGL_loadBitmapIntoDC* function.

See Also

MGL_unloadBitmap, *MGL_availableBitmap*, *MGL_getBitmapSize*, *MGL_loadBitmapIntoDC*,
MGL_saveBitmapFromDC, *MGL_putBitmap*, *MGL_loadBitmapExt*

MGL_loadBitmapExt

Locates an open bitmap and loads it into memory from an open file.

Declaration

```
bitmap_t * MGLAPI MGL_loadBitmapExt (
    FILE *f,
    ulong dwOffset,
    ulong dwSize,
    ibool loadPalette)
```

Prototype In

mgraph.h

Parameters

<i>f</i>	Open binary file to read data from
<i>dwOffset</i>	Offset to start of bitmap in file
<i>dwSize</i>	Size of the file
<i>loadPalette</i>	Should we load the palette values as well?

Return Value

Pointer to the loaded bitmap file

Description

This function is the same as *MGL_loadBitmap*, however it loads the file from a previously open file. This allows you to create your own large files with multiple files embedded in them.

See Also

MGL_loadBitmap, *MGL_loadBitmapIntoDC*

MGL_loadBitmapIntoDC

Loads a bitmap file directly into an existing device context.

Declaration

```
ibool MGLAPI MGL_loadBitmapIntoDC (
    MGLDC *dc,
    const char *bitmapName,
    int dstLeft,
    int dstTop,
    ibool loadPalette)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to load bitmap into
<i>bitmapName</i>	Name of bitmap file to load
<i>dstLeft</i>	Left coordinate to load bitmap at
<i>dstTop</i>	Top coordinate to load bitmap at
<i>loadPalette</i>	True if the palette should also be loaded

Return Value

True if the bitmap was loaded, false on error.

Description

Locates the specified bitmap file and loads it into the specified device context at the specified destination coordinates. If the bitmap is of a different pixel depth then the device context that it is being loaded into, the bitmap will be converted as it is loaded to the pixel format of the device context it is being loaded into. MGL can load any Windows 3.x style bitmap files, including new format bitmap files with colors depths of 15/16 and 32 bits per pixel. Consult the Windows SDK documentation for the format of Windows bitmap files.

If loadPalette is true, the palette values for the bitmap will be loaded and stored in the device context's palette. If the device context being loaded into is the currently active display device, the palette will also be realized before the bits in the bitmap are loaded.

When MGL is searching for bitmap files it will first attempt to find the files just by using the filename itself. Hence if you wish to look for a specific bitmap file, you should pass the full pathname to the file that you are interested in. If the filename is a simple relative filename (i.e. "MYBMP.BMP"), MGL will then search in the BITMAPS directory relative to the path specified in mglpath variable that was passed to *MGL_init*. As a final resort MGL will also look for the files in the BITMAPS directory relative to the MGL_ROOT environment variable.

If the bitmap file was not found, or an error occurred while reading the bitmap file, this function will return false. You can check the *MGL_result* error code to determine the cause.

See Also

MGL_availableBitmap, *MGL_getBitmapSize*, *MGL_loadBitmap*, *MGL_saveBitmapFromDC*, *MGL_loadBitmapIntoDCExt*

MGL_loadBitmapIntoDCExt

Locates the specified bitmap file and loads it into a device context.

Declaration

```
ibool MGLAPI MGL_loadBitmapIntoDCExt (
    MGLDC *dc,
    FILE *f,
    ulong dwOffset,
    ulong dwSize,
    int dstLeft,
    int dstTop,
    ibool loadPalette)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to load bitmap into
<i>f</i>	Pointer to opened bitmap file
<i>dwOffset</i>	Offset into the file
<i>dwSize</i>	Size of the bitmap file
<i>dstLeft</i>	Left coordinate to place bitmap at
<i>dstTop</i>	Top coordinate to place bitmap at
<i>loadPalette</i>	Should we load the palette values as well?

Return Value

True if the bitmap was successfully loaded.

Description

This function is the same as *MGL_loadBitmapIntoDC*, however it works with a previously opened file. This allows you to create your own large files with multiple files embedded in them.

See Also

MGL_loadBitmapIntoDC

MGL_loadCursor

Load a cursor file from disk.

Declaration

```
cursor_t * MGLAPI MGL_loadCursor(  
    const char *cursorName)
```

Prototype In

mgraph.h

Parameters

cursorName Name of cursor file to load

Return Value

Pointer to loaded cursor file, NULL on error.

Description

Locates the specified cursor file and loads it into memory. MGL can load any Windows 3.x style cursor files. Consult the Windows SDK documentation for the format of Windows cursor files.

When MGL is searching for cursor files it will first attempt to find the files just by using the filename itself. Hence if you wish to look for a specific cursor file, you should pass the full pathname to the file that you are interested in. If the filename is a simple relative filename (i.e. "MYCURS.CUR"), MGL will then search in the CURSORS directory relative to the path specified in *mgldpath* variable that was passed to *MGL_init*. As a final resort MGL will also look for the files in the CURSORS directory relative to the MGL_ROOT environment variable.

If the cursor file was not found, or an error occurred while reading the cursor file, this function will return NULL. You can check the *MGL_result* error code to determine the cause.

See Also

MGL_unloadCursor, *MGL_availableCursor*, *MS_setCursor*, *MGL_loadCursorExt*

MGL_loadCursorExt

Load a cursor file from disk from an opened file.

Declaration

```
cursor_t * MGLAPI MGL_loadCursorExt (
    FILE *f,
    ulong dwOffset,
    ulong dwSize)
```

Prototype In

mgraph.h

Parameters

<i>f</i>	Open file to read cursor from (binary mode)
<i>dwOffset</i>	Offset to the start of the cursor file
<i>dwSize</i>	Size of the file

Return Value

Pointer to the loaded cursor file

Description

This function is the same as *MGL_loadCursor*, however it works with a previously opened file. This allows you to create your own large files with multiple files embedded in them.

See Also

MGL_loadCursor

MGL_loadFont

Load a font file for use.

Declaration

```
font_t * MGLAPI MGL_loadFont(
    const char *fontname)
```

Prototype In

mgraph.h

Parameters

fontname Name of the font file to load

Return Value

Pointer to the loaded font file, NULL on error.

Description

Locates the specified font file and loads it into memory. MGL can load any Windows 2.x style font files (Windows 3.x font files are not supported, but Windows 2.x font files are the standard files even for Windows 3.1. Most resource editors can only output 2.x style font files). Consult the Windows SDK documentation for the format of Windows font files.

When MGL is searching for font files it will first attempt to find the files just by using the filename itself. Hence if you wish to look for a specific font file, you should pass the full pathname to the file that you are interested in. If the filename is a simple relative filename (i.e. "MYFONT.FNT"), MGL will then search in the FONTS directory relative to the path specified in mglpath variable that was passed to *MGL_init*. As a final resort MGL will also look for the files in the FONTS directory relative to the MGL_ROOT environment variable.

If the font file was not found, or an error occurred while reading the font file, this function will return NULL. You can check the *MGL_result* error code to determine the cause.

See Also

MGL_unloadFont, *MGL_useFont*, *MGL_availableFont*.

MGL_loadFontExt

Load a font file for use from an opened file.

Declaration

```
font_t * MGLAPI MGL_loadFontExt(  
    FILE *f,  
    ulong dwOffset,  
    ulong dwSize)
```

Prototype In

mgraph.h

Parameters

<i>f</i>	Open binary file to read font data from
<i>dwOffset</i>	Offset to start of font in file
<i>dwSize</i>	Size of the file in bytes

Return Value

Pointer to the font data, or NULL on error.

Description

This function is the same as *MGL_loadFont*, however it works with a previously opened file. This allows you to create your own large files with multiple files embedded in them.

See Also

MGL_loadFont

MGL_loadFontInstance

Load a font file for use.

Declaration

```
font_t * MGLAPI MGL_loadFontInstance(
    font_lib_t *fontlib,
    float pointSize,
    float slant,
    float angle,
    ibool antialiased)
```

Prototype In

mgraph.h

Return Value

Pointer to the loaded font file, NULL on error.

Description

Locates the specified font file and loads it into memory. MGL can load any Windows 2.x style font files (Windows 3.x font files are not supported, but Windows 2.x font files are the standard files even for Windows 3.1. Most resource editors can only output 2.x style font files). Consult the Windows SDK documentation for the format of Windows font files.

When MGL is searching for font files it will first attempt to find the files just by using the filename itself. Hence if you wish to look for a specific font file, you should pass the full pathname to the file that you are interested in. If the filename is a simple relative filename (i.e. "MYFONT.FNT"), MGL will then search in the FONTS directory relative to the path specified in mglpath variable that was passed to *MGL_init*. As a final resort MGL will also look for the files in the FONTS directory relative to the MGL_ROOT environment variable.

If the font file was not found, or an error occurred while reading the font file, this function will return NULL. You can check the *MGL_result* error code to determine the cause.

See Also

MGL_openFontLib, *MGL_unloadFontInstance*, *MGL_closeFontLib*

MGL_loadIcon

Load an icon file from disk.

Declaration

```
icon_t * MGLAPI MGL_loadIcon(
    const char *iconName,
    ibool loadPalette)
```

Prototype In

mgraph.h

Parameters

<i>iconName</i>	Name of icon file to load
<i>loadPalette</i>	True if the palette should also be loaded

Return Value

Pointer to the loaded icon file, NULL on error.

Description

Locates the specified icon file and loads it into memory. MGL can load any Windows 3.x style icon files of any dimensions. Generally icon files will be either 32x32 or 64x64 in size. Consult the Windows SDK documentation for the format of Windows font files.

If loadPalette is true, the palette values for the icon will be loaded into the structure as well (if there is no palette, it will not be loaded), otherwise the palette entry for the bitmap will be NULL.

When MGL is searching for icon files it will first attempt to find the files just by using the filename itself. Hence if you wish to look for a specific icon file, you should pass the full pathname to the file that you are interested in. If the filename is a simple relative filename (i.e. "MYICON.ICO"), MGL will then search in the ICONS directory relative to the path specified in mglpath variable that was passed to *MGL_init*. As a final resort MGL will also look for the files in the ICONS directory relative to the MGL_ROOT environment variable.

If the icon file was not found, or an error occurred while reading the icon file, this function will return NULL. You can check the *MGL_result* error code to determine the cause.

See Also

MGL_unloadIcon, *MGL_availableIcon*, *MGL_putIcon*, *MGL_loadIconExt*

MGL_loadIconExt

Load an icon file from disk from an open file.

Declaration

```
icon_t * MGLAPI MGL_loadIconExt(
    FILE *f,
    ulong dwOffset,
    ulong dwSize,
    ibool loadPalette)
```

Prototype In

mgraph.h

Parameters

<i>f</i>	Pointer to open icon file
<i>dwOffset</i>	Offset into the icon file
<i>dwSize</i>	Size of the icon file
<i>loadPalette</i>	If true, the palette is loaded as well

Return Value

Pointer to the loaded icon file.

Description

This function is the same as *MGL_loadIcon*, however it works with a previously opened file. This allows you to create your own large files with multiple files embedded in them.

See Also

MGL_loadIcon

MGL_loadJPEG

Load a JPEG bitmap file from disk.

Declaration

```
bitmap_t * MGLAPI MGL_loadJPEG(
    const char *JPEGName,
    int num8BitColors)
```

Prototype In

mgraph.h

Parameters

<i>JPEGName</i>	Name of JPEG file to load
<i>num8BitColors</i>	Number of colors for 8-bit image, 0 for RGB images, -1 for grayscale

Return Value

Pointer to the loaded JPEG file, NULL on error.

Description

Locates the specified JPEG file and loads it into a lightweight bitmap structure. Because JPEG files are inherently 24-bit, when you load a JPEG file with this function it will always be decoded as a 24-bit RGB bitmap file, unless you set num8BitColors parameter. If the num8BitColors parameter is set to a value other than 0, it causes the JPEG decoder to quantize down to an 8 bits per pixel bitmap with an optimized floyd-steinberg dither (better than the MGL's simple halftone dithering, but the bitmap will contain a custom palette). The number of significant colors in the output image will be set to the value you specify, so you can use this to quantise down to a color table smaller than 8-bits per pixel. To decode as a 24-bit image, simply set this field to 0.

Note that if you set num8BitColors to -1, the JPEG decoder will decode the image as a grayscale bitmap which is faster than decoding the full color image (useful for preview operations etc). Note that images that are decoded as grayscale are 8-bits per pixel with a grayscale color map and should display as true grayscale images in all color depths.

If you wish to load the bitmap as a different color depth or pixel format use the *MGL_loadJPEGIntoDC* function. The MGL will however properly decode grayscale JPEG files, but they will be loaded as a 24-bit bitmap since the MGL does not natively support grayscale bitmaps.

When MGL is searching for JPEG files it will first attempt to find the files just by using the filename itself. Hence if you wish to look for a specific JPEG file, you should pass the full pathname to the file that you are interested in. If the filename is a simple relative filename (i.e. "MYBMP.JPG"), MGL will then search in the BITMAPS directory relative to the path specified in mglpath variable that was passed to *MGL_init*. As a final resort MGL will also look for the files in the BITMAPS directory relative to the MGL_ROOT environment variable.

If the JPEG file was not found, or an error occurred while reading the JPEG file, this function will return NULL. You can check the *MGL_result* error code to determine the cause.

The routine allocates a lightweight bitmap structure for holding the JPEG file, which loads the bitmap with the minimum memory overheads. You can draw the JPEG file on any device context surface using the *MGL_putBitmap* function, but you don't have the full flexibility of using a memory device context for the bitmap surface. If you need more control over the bitmap, you can allocate a memory device context to hold the bitmap data and load the bitmap with the *MGL_loadJPEGIntoDC* function.

See Also

MGL_unloadBitmap, *MGL_availableJPEG*, *MGL_getJPEGSize*, *MGL_loadJPEGIntoDC*, *MGL_saveJPEGFromDC*, *MGL_putBitmap*, *MGL_loadJPEGExt*

MGL_loadJPEGExt

Load a JPEG bitmap file from disk using an open file.

Declaration

```
bitmap_t * MGLAPI MGL_loadJPEGExt(
    FILE *f,
    ulong dwOffset,
    ulong dwSize,
    int num8BitColors)
```

Prototype In

mgraph.h

Parameters

<i>f</i>	Open binary file to read bitmap from
<i>dwOffset</i>	Offset to start of JPEG file within open file
<i>dwSize</i>	Size of JPEG file in bytes
<i>num8BitColors</i>	Number of colors for 8-bit image, 0 for RGB images, -1 for grayscale

Return Value

Pointer to the loaded bitmap file

Description

This function is the same as *MGL_loadJPEG*, however it loads the file from a previously open file. This allows you to create your own large files with multiple files embedded in them.

See Also

MGL_loadJPEG, *MGL_loadBitmap*

MGL_loadJPEGIntoDC

Loads a JPEG file directly into an existing device context.

Declaration

```
ibool MGLAPI MGL_loadJPEGIntoDC(
    MGLDC *dc,
    const char *JPEGName,
    int dstLeft,
    int dstTop,
    int num8BitColors)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to load bitmap into
<i>JPEGName</i>	Name of JPEG file to load
<i>dstLeft</i>	Left coordinate to load JPEG at
<i>dstTop</i>	Top coordinate to load JPEG at
<i>num8BitColors</i>	Number of colors for 8-bit image, 0 for RGB images, -1 for grayscale

Return Value

True if the JPEG file was loaded, false on error.

Description

Locates the specified JPEG file and loads it into the specified device context at the specified destination coordinates. If the JPEG is of a different pixel depth than the device context that it is being loaded into, the JPEG will be converted as it is loaded to the pixel format of the device context it is being loaded into.

If the num8BitColors parameter is set to a value other than 0, it causes the JPEG decoder to quantize down to an 8 bits per pixel bitmap with an optimized floyd-steenberg dither (better than the MGL's simple halftone dithering) using the palette of the destination device context as the source for the dither. The number of significant colors used from the destination device contexts palette for the dither is set by the num8BitColors parameter. If however the destination device context is not 8-bits per pixel, the image is simply decoded as a 24-bit bitmap.

Note that if you set num8BitColors to -1, the JPEG decoder will decode the image as a grayscale bitmap which is faster than decoding the full color image (useful for preview operations etc). Note that images that are decoded as grayscale are 8-bits per pixel with a grayscale color map and should display as true grayscale images in all color depths. If the destination device context is 8-bits per pixel, the color palette will be changed to a grayscale color palette.

When MGL is searching for bitmap files it will first attempt to find the files just by using the filename itself. Hence if you wish to look for a specific bitmap file, you should pass the full pathname to the file that you are interested in. If the filename is a simple relative filename (i.e. "MYFILE.JPEG"), MGL will then search in the BITMAPS directory relative to the path specified in *mgldpath* variable that was passed to *MGL_init*. As a final resort MGL will also look for the files in the BITMAPS directory relative to the MGL_ROOT environment variable.

If the bitmap file was not found, or an error occurred while reading the bitmap file, this function will return false. You can check the *MGL_result* error code to determine the cause.

See Also

MGL_availableJPEG, MGL_getJPEGSize, MGL_loadJPEG, MGL_saveJPEGFromDC

MGL_loadJPEGIntoDCExt

Loads a JPEG file directly into an existing device context using an open file.

Declaration

```
ibool MGLAPI MGL_loadJPEGIntoDCExt (
    MGLDC *dc,
    FILE *f,
    ulong dwOffset,
    ulong dwSize,
    int dstLeft,
    int dstTop,
    int num8BitColors)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to load bitmap into
<i>f</i>	Open binary file to read bitmap from
<i>dwOffset</i>	Offset to start of JPEG file within open file
<i>dwSize</i>	Size of JPEG file in bytes
<i>dstLeft</i>	Left coordinate to load JPEG at
<i>dstTop</i>	Top coordinate to load JPEG at
<i>num8BitColors</i>	Number of colors for 8-bit image, 0 for RGB images, -1 for grayscale

Return Value

Pointer to the loaded JPEG file, NULL on error.

Description

This function is the same as *MGL_loadJPEGIntoDC*, however it loads the file from a previously open file. This allows you to create your own large files with multiple files embedded in them.

See Also

MGL_loadJPEGIntoDC, *MGL_loadBitmapIntoDC*

MGL_loadPCX

Load a lightweight PCX file from disk.

Declaration

```
bitmap_t * MGLAPI MGL_loadPCX(
    const char *PCXName,
    ibool loadPalette)
```

Prototype In

mgraph.h

Parameters

<i>PCXName</i>	Name of PCX file to load
<i>loadPalette</i>	True if the palette should also be loaded

Return Value

Pointer to the loaded PCX file, NULL on error.

Description

Locates the specified PCX file and loads it into a lightweight bitmap structure. MGL can load any 1, 4, 8 or 24 bits per pixel PCX file.

If *loadPalette* is true, the palette values for the PCX will be loaded into the structure as well (if there is no palette, it will not be loaded), otherwise the palette entry for the PCX will be NULL. For small PCX files you can save space by not loading the palette for the PCX.

When MGL is searching for PCX files it will first attempt to find the files just by using the filename itself. Hence if you wish to look for a specific PCX file, you should pass the full pathname to the file that you are interested in. If the filename is a simple relative filename (i.e. "MYBMP.PCX"), MGL will then search in the BITMAPS directory relative to the path specified in *mgldpath* variable that was passed to *MGL_init*. As a final resort MGL will also look for the files in the BITMAPS directory relative to the MGL_ROOT environment variable.

If the PCX file was not found, or an error occurred while reading the PCX file, this function will return NULL. You can check the *MGL_result* error code to determine the cause.

The routine allocates a lightweight bitmap structure for holding the PCX file, which loads the bitmap with the minimum memory overheads. You can draw the PCX file on any device context surface using the *MGL_putBitmap* function, but you don't have the full flexibility of using a memory device context for the bitmap surface. If you need more control over the bitmap, you can allocate a memory device context to hold the bitmap data and load the bitmap with the *MGL_loadPCXIntoDC* function.

See Also

*MGL_unloadBitmap, MGL_availablePCX, MGL_getPCXSize, MGL_loadPCXIntoDC,
MGL_savePCXFromDC, MGL_putBitmap, MGL_loadPCXExt*

MGL_loadPCXExt

Load a lightweight PCX file from disk using an open file.

Declaration

```
bitmap_t * MGLAPI MGL_loadPCXExt(
    FILE *f,
    ulong dwOffset,
    ulong dwSize,
    ibool loadPalette)
```

Prototype In

mgraph.h

Parameters

<i>f</i>	Open binary file to read bitmap from
<i>dwOffset</i>	Offset to start of PCX file within open file
<i>dwSize</i>	Size of PCX file in bytes
<i>loadPalette</i>	If true, palette values are loaded as well.

Return Value

Pointer to the loaded bitmap file

Description

This function is the same as *MGL_loadPCX*, however it loads the file from a previously open file. This allows you to create your own large files with multiple files embedded in them.

See Also

MGL_loadPCX, *MGL_loadBitmap*

MGL_loadPCXIntoDC

Loads a PCX file directly into an existing device context.

Declaration

```
ibool MGLAPI MGL_loadPCXIntoDC(
    MGLDC *dc,
    const char *PCXName,
    int dstLeft,
    int dstTop,
    ibool loadPalette)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to load bitmap into
<i>PCXName</i>	Name of PCX file to load
<i>dstLeft</i>	Left coordinate to load PCX at
<i>dstTop</i>	Top coordinate to load PCX at
<i>loadPalette</i>	True if the palette should also be loaded

Return Value

True if the PCX file was loaded, false on error.

Description

Locates the specified PCX file and loads it into the specified device context at the specified destination coordinates. If the PCX is of a different pixel depth than the device context that it is being loaded into, the PCX will be converted as it is loaded to the pixel format of the device context it is being loaded into. MGL can load any 1, 4, 8 or 24 bits per pixel PCX file.

If loadPalette is true, the palette values for the PCX will be loaded and stored in the device context's palette. If the device context being loaded into is the currently active display device, the palette will also be realized before the bits in the bitmap are loaded.

When MGL is searching for bitmap files it will first attempt to find the files just by using the filename itself. Hence if you wish to look for a specific bitmap file, you should pass the full pathname to the file that you are interested in. If the filename is a simple relative filename (i.e. "MYFILE.PCX"), MGL will then search in the BITMAPS directory relative to the path specified in mglpath variable that was passed to *MGL_init*. As a final resort MGL will also look for the files in the BITMAPS directory relative to the MGL_ROOT environment variable.

If the bitmap file was not found, or an error occurred while reading the bitmap file, this function will return false. You can check the *MGL_result* error code to determine the cause.

See Also

MGL_availablePCX, MGL_getPCXSize, MGL_loadPCX, MGL_savePCXFromDC

MGL_loadPCXIntoDCExt

Load a lightweight PCX file from disk using an open file.

Declaration

```
ibool MGLAPI MGL_loadPCXIntoDCExt(
    MGLDC *dc,
    FILE *f,
    ulong dwOffset,
    ulong dwSize,
    int dstLeft,
    int dstTop,
    ibool loadPalette)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to load bitmap into
<i>f</i>	Open binary file to read bitmap from
<i>dwOffset</i>	Offset to start of PCX file within open file
<i>dwSize</i>	Size of PCX file in bytes
<i>dstLeft</i>	Left coordinate to load PCX at
<i>dstTop</i>	Top coordinate to load PCX at
<i>loadPalette</i>	True if the palette should also be loaded

Return Value

Pointer to the loaded PCX file, NULL on error.

Description

This function is the same as *MGL_loadPCXIntoDC*, however it loads the file from a previously open file. This allows you to create your own large files with multiple files embedded in them.

See Also

MGL_loadPCXIntoDC, *MGL_loadBitmapIntoDC*

MGL_loadPNG

Load a PNG bitmap file from disk.

Declaration

```
bitmap_t * MGLAPI MGL_loadPNG(
    const char *PNGName,
    ibool loadPalette)
```

Prototype In

mgraph.h

Parameters

<i>PNGName</i>	Name of PNG file to load
<i>loadPalette</i>	True if you wish to load the images palette Only valid on PNG files with a bit depth of 8 or below

Return Value

Pointer to the loaded PNG file, NULL on error.

Description

If you wish to load the bitmap as a different color depth or pixel format use the *MGL_loadPNGIntoDC* function.

When MGL is searching for PNG files it will first attempt to find the files just by using the filename itself. Hence if you wish to look for a specific PNG file, you should pass the full pathname to the file that you are interested in. If the filename is a simple relative filename (i.e. "MYBMP.PNG"), MGL will then search in the BITMAPS directory relative to the path specified in mglpath variable that was passed to *MGL_init*. As a final resort MGL will also look for the files in the BITMAPS directory relative to the MGL_ROOT environment variable.

If the PNG file was not found, or an error occurred while reading the PNG file, this function will return NULL. You can check the *MGL_result* error code to determine the cause.

See Also

MGL_unloadBitmap, *MGL_availablePNG*, *MGL_getPNGSize*, *MGL_loadPNGIntoDC*, *MGL_savePNGFromDC*, *MGL_putBitmap*, *MGL_loadPNGExt*

MGL_loadPNGExt

Load a PNG bitmap file from disk using an open file.

Declaration

```
bitmap_t * MGLAPI MGL_loadPNGExt(
    FILE *f,
    ulong dwOffset,
    ulong dwSize,
    ibool loadPalette)
```

Prototype In

mgraph.h

Parameters

<i>f</i>	Open binary file to read bitmap from
<i>dwOffset</i>	Offset to start of PNG file within open file
<i>dwSize</i>	Size of PNG file in bytes
<i>loadPalette</i>	true if you wish to load the images palette

Return Value

Pointer to the loaded bitmap file

Description

This function is the same as *MGL_loadPNG*, however it loads the file from a previously open file. This allows you to create your own large files with multiple files embedded in them.

See Also

MGL_loadPNG, *MGL_loadBitmap*

MGL_loadPNGIntoDC

Loads a PNG file directly into an existing device context.

Declaration

```
ibool MGLAPI MGL_loadPNGIntoDC(
    MGLDC *dc,
    const char *PNGName,
    int dstLeft,
    int dstTop,
    ibool loadPalette)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to load bitmap into
<i>PNGName</i>	Name of PNG file to load
<i>dstLeft</i>	Left coordinate to load PNG at
<i>dstTop</i>	Top coordinate to load PNG at
<i>loadPalette</i>	True if you would like to replace the dc's palette with PNG file's

Return Value

True if the PNG file was loaded, false on error.

Description

Locates the specified PNG file and loads it into the specified device context at the specified destination coordinates. If the PNG is of a different pixel depth than the device context that it is being loaded into, the PNG will be converted as it is loaded to the pixel format of the device context it is being loaded into.

If the PNG file has a higher bit depth than the surface the pixel values will be dithered to the correct color depth. This will use the currently selected palette if loadPalette is false, or the HalfTone palette if true.

When MGL is searching for bitmap files it will first attempt to find the files just by using the filename itself. Hence if you wish to look for a specific bitmap file, you should pass the full pathname to the file that you are interested in. If the filename is a simple relative filename (i.e. "MYFILE.PNG"), MGL will then search in the BITMAPS directory relative to the path specified in mglpath variable that was passed to *MGL_init*. As a final resort MGL will also look for the files in the BITMAPS directory relative to the MGL_ROOT environment variable.

If the bitmap file was not found, or an error occurred while reading the bitmap file, this function will return false. You can check the *MGL_result* error code to determine the cause.

If the PNG file has interlacing enabled MGL must create a memory buffer internally to render the image into before blitting to the dc. Therefore using non-interlaced images will reduce memory overhead and speed performance.

See Also

MGL_availablePNG, MGL_getPNGSize, MGL_loadPNG, MGL_savePNGFromDC

MGL_loadPNGIntoDCExt

Load a PNG file directly into an existing device context

Declaration

```
ibool MGLAPI MGL_loadPNGIntoDCExt (
    MGLDC *dc,
    FILE *f,
    ulong dwOffset,
    ulong dwSize,
    int dstLeft,
    int dstTop,
    ibool loadPalette)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to output to
<i>f</i>	Open binary file to read bitmap from
<i>dwOffset</i>	Offset to start of PNG file within open file
<i>dwSize</i>	Size of PNG file in bytes
<i>dstLeft</i>	Left coordinate to align left edge of bitmap with
<i>dstTop</i>	Top coordinate to align top edge of bitmap with
<i>loadPalette</i>	true if you wish to load the images palette

Return Value

True if the PNG was loaded, false on error.

Description

This function is the same as *MGL_loadPNGIntoDC*, however it loads the file from a previously open file. This allows you to create your own large files with multiple files embedded in them.

If the PNG file has interlacing enabled MGL must create a memory buffer internally to render the image into before blitting to the dc. Therefore using non-interlaced images will reduce memory overhead and speed performance.

See Also

MGL_loadPNGIntoDC, *MGL_loadBitmapIntoDCExt*, *MGL_loadPNG*

MGL_localToGlobal

Converts a point from local to global coordinates.

Declaration

```
void MGLAPI MGL_localToGlobal(  
    point_t *p)
```

Prototype In

mgraph.h

Parameters

p Pointer to point to be converted

Description

This function converts a coordinate from local coordinates to global coordinates. Global coordinates are defined relative to the entire output device display, while local coordinates are relative to the currently active viewport.

See Also

MGL_localToGlobalDC, *MGL_globalToLocal*

MGL_localToGlobalDC

Converts a point from local to global coordinates for a specific DC.

Declaration

```
void MGLAPI MGL_localToGlobalDC(  
    MGLDC *dc,  
    point_t *p)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context in which the point is defined
<i>p</i>	Pointer to point to be converted

Description

This function is the same as *MGL_localToGlobal*, however the device context does not have to be the current device context.

See Also

MGL_localToGlobal, *MGL_globalToLocal*

MGL_lockBuffer

Lock a buffer to begin direct surface access

Declaration

```
ulong MGLAPI MGL_lockBuffer(  
    MGLBUF *buf)
```

Prototype In

mgraph.h

Parameters

buf MGL 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!

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

MGL_unlockBuffer

MGL_lockToFrameRate

Block until a specific time has elapsed since the last call

Declaration

```
void MGLAPI MGL_lockToFrameRate(  
    ulong milliseconds)
```

Prototype In

mgraph.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.

MGL_makeCurrentDC

Make a device context the current device context.

Declaration

```
MGLDC * MGLAPI MGL_makeCurrentDC (
    MGLDC *dc)
```

Prototype In
mgraph.h

Parameters

dc New device context to make the current context

Return Value

Previous current device context (possibly NULL).

Description

This function makes the specified device context the current device context. The current device context is the one that is used for all the output rasterizing routines, and when a device context is made current, a copy of the device context is cached internally for maximum speed in the rasterizing code. While a device context is current, changes made to the global state of the current device context are changed in the cached copy only and are not updated in the original device context. When the device context is changed however, the values in the cached device context are flushed back to the original device context to keep it up to date. You can flush the current device context explicitly by passing a NULL for the new current device context.

Because of this caching mechanism, changing the current device context is an expensive operation, so you should try to minimize the need to change the current device context. Normally you will maintain a single device context that will be used for all rasterizing operations, and leave this as your current device context. If the device context specified is already the current device context, this function simply does nothing.

Note: *If the SciTech MGL has been initialised to support multiple display controllers, calling this function will ensure that the display device being accessed by the passed in device context is made the currently active display. If the display controllers can support 'mixed' mode, the secondary displays remain active at the same time as the primary display and there is no overhead for the switch. Some older display controllers cannot support mixed mode as the VGA compatible resources cannot be disabled, and in this case every call to this function will cause the active display device to be switched.*

See Also

MGL_isCurrentDC, MGL_initMultiMonitor

MGL_mapToPalette

Map the colors of a memory device context to match a new palette.

Declaration

```
void MGLAPI MGL_mapToPalette(  
    MGLDC *dc,  
    palette_t *pal)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Memory device context to map (8 bits per pixel only)
<i>pal</i>	New palette to map to

Description

This function maps the pixels of an 8 bits per pixel memory device context to the specified palette, and then sets the palette for the device context to the new palette. This function actually translates every pixel in the device context's surface to the new palette, by looking for the entry in the new palette that is the closest to color of the original pixel in the old palette (the one currently active before this routine was called).

See Also

MGL_setPalette

MGL_maxCharWidth

Returns the maximum character width for current font.

Declaration

```
int MGLAPI MGL_maxCharWidth(void)
```

Prototype In

mgraph.h

Return Value

Maximum character width for current font.

Description

Returns the maximum character width for the currently active font. You can use this routine to quickly determine if a character will possibly overlap something else on the device surface.

See Also

MGL_getCharMetrics, *MGL_getFontMetrics*, *MGL_textHeight*, *MGL_textWidth*, *MGL_charWidth*

MGL_maxColor

Returns the maximum available color value.

Declaration

```
color_t MGLAPI MGL_maxColor(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context to check

Return Value

Maximum color value for current video mode.

Description

Returns the value of the largest available color value for the current video mode. This value will always be one less than the number of available colors in that particular video mode.

MGL_maxPage

Returns the maximum available hardware video page index.

Declaration

```
int MGLAPI MGL_maxPage(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context to check.

Return Value

Index of maximum available hardware video page.

Description

Returns the index of the highest hardware video page that is available. This value will always be one less than the number of hardware video pages available. Some video modes only have one hardware video page available, so this value will be 0.

MGL_maxx

Returns the current maximum x coordinate.

Declaration

```
int MGLAPI MGL_maxx(void)
```

Prototype In

mgraph.h

Return Value

Maximum x coordinate in current viewport.

Description

Returns the maximum x coordinate available in the currently active viewport. This value will change if you change the dimensions of the current viewport.

Use the *MGL_sizex* routine to determine the dimensions of the physical display area available to the application.

See Also

MGL_maxxDC, *MGL_maxy*, *MGL_sizex*, *MGL_sizey*

MGL_maxxDC

Returns the current maximum x coordinate for a specific DC.

Declaration

```
int MGLAPI MGL_maxxDC(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context of the target x coordinate

Return Value

Maximum x coordinate in current viewport.

Description

This function is the same as *MGL_maxx*, however the device context does not have to be the current device context.

See Also

MGL_maxx, *MGL_maxy*, *MGL_sizeX*, *MGL_sizeY*

MGL_maxy

Returns the current maximum y coordinate.

Declaration

```
int MGLAPI MGL_maxy(void)
```

Prototype In

mgraph.h

Return Value

Maximum y coordinate in current viewport.

Description

Returns the maximum y coordinate available in the currently active viewport. This value will change if you change the dimensions of the current viewport.

Use the *MGL_sizey* routine to determine the dimensions of the physical display area available to the program.

See Also

MGL_maxyDC, *MGL_maxx*, *MGL_sizex*, *MGL_sizey*

MGL_maxyDC

Returns the current maximum y coordinate for a specific DC.

Declaration

```
int MGLAPI MGL_maxyDC(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context of the target y coordinate

Return Value

Maximum y coordinate in current viewport.

Description

This function is the same as *MGL_maxy*, however the device context does not have to be the current device context.

See Also

MGL_maxy, *MGL_maxx*, *MGL_sizex*, *MGL_sizey*

MGL_memcpy

Copies a block of memory as fast as possible.

Declaration

```
void MGLAPI MGL_memcpy(  
    void *dst,  
    void *src,  
    int n)
```

Prototype In

mgraph.h

Parameters

<i>dst</i>	Pointer to destination block
<i>src</i>	Pointer to source block
<i>n</i>	Number of bytes to copy

Description

This function copies a block of memory as fast as possible, and has been optimized to copy the data 32 bits at a time for maximum performance. This function is similar to the standard C library memcpy function, but can correctly handle copying of memory blocks that are larger than 64Kb in size for 16 bit real mode environments. Note also that this function is generally a lot faster than some standard C library functions.

See Also

MGL_memcpyVIRTSRC, *MGL_memcpyVIRTDST*

MGL_memcpyVIRTDST

Copies a block of memory as fast as possible.

Declaration

```
void MGLAPI MGL_memcpyVIRTDST(  
    void *dst,  
    void *src,  
    int n)
```

Prototype In mgraph.h

Parameters

<i>dst</i>	Pointer to destination block
<i>src</i>	Pointer to source block
<i>n</i>	Number of bytes to copy

Description

This function copies a block of memory as fast as possible, and has been optimized to copy the data 32 bits at a time for maximum performance. This function is similar to the standard C library `memcpy` function, but can correctly handle copying of memory blocks that are larger than 64Kb in size for 16 bit real mode environments. Note also that this function is generally a lot faster than some standard C library functions.

This function is identical to *MGL_memcpy* except that it is virtual linear framebuffer safe, and should be used for copying data where the destination pointer resides in a virtualized linear surface.

See Also

MGL_memcpyVIRT SRC, *MGL_memcpy*

MGL_memcpyVIRTSRC

Copies a block of memory as fast as possible.

Declaration

```
void MGLAPI MGL_memcpyVIRTSRC(
    void *dst,
    void *src,
    int n)
```

Prototype In mgraph.h

Parameters

<i>dst</i>	Pointer to destination block
<i>src</i>	Pointer to source block
<i>n</i>	Number of bytes to copy

Description

This function copies a block of memory as fast as possible, and has been optimized to copy the data 32 bits at a time for maximum performance. This function is similar to the standard C library memcpy function, but can correctly handle copying of memory blocks that are larger than 64Kb in size for 16 bit real mode environments. Note also that this function is generally a lot faster than some standard C library functions.

This function is identical to *MGL_memcpy* except that it is virtual linear framebuffer safe, and should be used for copying data where the source pointer resides in a virtualized linear surface.

See Also

MGL_memcpy, *MGL_memcpyVIRTDEST*

MGL_memset

Clears a memory block with 8-bit values.

Declaration

```
void MGLAPI MGL_memset(  
    void *p,  
    int c,  
    long n)
```

Prototype In mgraph.h

Parameters

<i>p</i>	Pointer to block to clear
<i>c</i>	Value to clear with
<i>n</i>	Number of bytes to clear

Description

This function clears a memory block to the specified 8 bit value. This function is similar to the standard C library `memset` function, but can correctly handle clearing of memory blocks that are larger than 64Kb in size for 16 bit real mode environments.

See Also

MGL_memsetw, *MGL_memsetl*

MGL_memsetl

Clears a memory block with 32-bit values.

Declaration

```
void MGLAPI MGL_memsetl(  
    void *p,  
    long c,  
    long n)
```

Prototype In

mgraph.h

Parameters

<i>p</i>	Pointer to block to clear
<i>c</i>	Value to clear with
<i>n</i>	Number of dwords to clear

Description

This function clears a memory block to the specified 32 bit value. This function is similar to the standard C library `memset` function, but can correctly handle clearing of memory blocks that are larger than 64Kb in size for 16 bit real mode environments, and allows you to specify a specific 32 bit value to be cleared.

See Also

MGL_memset, *MGL_memsetw*

MGL_memsetw

Clears a memory block with 16-bit values.

Declaration

```
void MGLAPI MGL_memsetw(  
    void *p,  
    int c,  
    long n)
```

Prototype In

mgraph.h

Parameters

<i>p</i>	Pointer to block to clear
<i>c</i>	Value to clear with
<i>n</i>	Number of words to clear

Description

This function clears a memory block to the specified 16 bit value. This function is similar to the standard C library `memset` function, but can correctly handle clearing of memory blocks that are larger than 64Kb in size for 16 bit real mode environments, and allows you to specify a specific 16 bit value to be cleared.

See Also

MGL_memset, *MGL_memsetl*

MGL_mirrorGlyph

Computes mirror image of a glyph (monochrome bitmap).

Declaration

```
void MGLAPI MGL_mirrorGlyph(  
    uchar *dst,  
    uchar *src,  
    int byteWidth,  
    int height)
```

Prototype In

mgraph.h

Parameters

<i>dst</i>	Destination glyph buffer
<i>src</i>	Source glyph buffer
<i>byteWidth</i>	Width of the glyph in bytes
<i>height</i>	Height of the glyph in scanlines

Description

This function computes the mirror image glyph of the source glyph, and stores the value in the destination buffer. The source buffer is not modified, and the mirror image glyph will be no larger than the original glyph.

See Also

MGL_rotateGlyph, *MGL_drawGlyph*

MGL_modeDriverName

Get a the name of the device driver used for particular graphics mode.

Declaration

```
const char * MGLAPI MGL_modeDriverName(  
    int mode)
```

Prototype In

mgraph.h

Parameters

mode graphics mode number

Return Value

Pointer to the name of device driver serving this mode

Description

This function returns the name of the device driver that is currently being used to support the specified graphics mode. The MGL provides a number of device drivers for supporting the fullscreen graphics mode resolutions depending on the capabilities of the underlying hardware. This function allows you to determine which driver is currently being used to support each mode.

MGL_modeFlags

Returns the mode flags for the specific graphics mode.

Declaration

```
ulong MGLAPI MGL_modeFlags (
    int mode)
```

Prototype In

mgraph.h

Parameters

mode graphics mode number to get flags for

Return Value

Flags for the specific graphics mode, or 0 for invalid mode.

Description

This function returns mode flags for the specified mode. The mode flags are available after calling *MGL_init*, and provides information about the hardware capabilities of the graphics mode, such as whether it supports 2D or 3D acceleration, video acceleration, refresh rate control and hardware stereo support. You can use these flags to make choices about the graphics mode to use for your application prior to initialising a specific graphics mode.

Specific mode flags are enumerated in *MGL_modeFlagsType*.

Note: *If the hardware has not been detected when this call is made, the MGL will automatically detect the installed hardware the first time this function is called.*

See Also

MGL_init, *MGL_availablePages*, *MGL_modeResolution*, *MGL_findMode*,
MGL_addCustomMode, *MGL_createDisplayDC*

MGL_modeResolution

Returns the resolution and pixel depth of a specific graphics mode.

Declaration

```
ibool MGLAPI MGL_modeResolution(
    int mode,
    int *xRes,
    int *yRes,
    int *bitsPerPixel)
```

Prototype In

mgraph.h

Parameters

<i>mode</i>	graphics mode number to get resolution for
<i>xRes</i>	Place to store the x resolution
<i>yRes</i>	Place to store the y resolution
<i>bitsPerPixel</i>	Place to store the pixel depth

Return Value

True on success, false for an invalid graphics mode number.

Description

This function returns the resolution and color depth of the specified MGL graphics mode. This function is useful for displaying the list of available modes to the user, or to search for specific graphics modes for use by an application program depending on the desired resolution or color depth.

Note: *If the hardware has not been detected when this call is made, the MGL will automatically detect the installed hardware the first time this function is called.*

See Also

MGL_init, MGL_availablePages, MGL_findMode, MGL_addCustomMode, MGL_createDisplayDC

MGL_moveRel

Moves the CP to a new relative location.

Declaration

```
void MGL_moveRel(  
    point_t p)
```

Prototype In

mgraph.h

Parameters

p Use coordinates of this point as offsets

Description

This function is the same as *MGL_moveRelCoord*, however it takes the amount to move as a point.

See Also

MGL_moveRelCoord

MGL_moveRelCoord

Moves the CP to a new relative location.

Declaration

```
void MGLAPI MGL_moveRelCoord(  
    int dx,  
    int dy)
```

Prototype In

mgraph.h

Parameters

<i>dx</i>	Amount to move x coordinate
<i>dy</i>	Amount to move y coordinate

Description

Moves the current position (CP) to the relative location that is a distance of (dx,dy) away from the CP. Thus the location the CP is moved to is (CP.x + dx, CP.y + dy).

See Also

MGL_moveRel

MGL_moveTo

Moves the CP to a new location.

Declaration

```
void MGL_moveTo(  
    point_t p)
```

Prototype In

mgraph.h

Parameters

p New Point for CP

Description

This function is the same as *MGL_moveToCoord*, however it takes the coordinate to move to as a point.

See Also

MGL_moveToCoord

MGL_moveToCoord

Moves the CP to a new location.

Declaration

```
void MGLAPI MGL_moveToCoord(  
    int x,  
    int y)
```

Prototype In

mgraph.h

Parameters

<i>x</i>	New x coordinate for CP
<i>y</i>	New y coordinate for CP

Description

Moves the current position (CP) to the new point (x,y).

See Also

MGL_moveTo

MGL_newRegion

Allocate a new complex region.

Declaration

```
region_t * MGLAPI MGL_newRegion(void)
```

Prototype In

mgraph.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 MGL 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

MGL_freeRegion, MGL_unionRegion, MGL_diffRegion, MGL_sectRegion

MGL_offsetRect

Offsets a rectangle by the specified amount.

Declaration

```
void MGL_offsetRect(  
    rect_t r,  
    int dx,  
    int dy)
```

Prototype In

mgraph.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

MGL_insetRect

MGL_offsetRegion

Offsets a region by the specified amount.

Declaration

```
void MGLAPI MGL_offsetRegion(  
    region_t *r,  
    int dx,  
    int dy)
```

Prototype In

mgraph.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

MGL_unionRegion, MGL_diffRegion, MGL_sectRegion

MGL_openFontLib

Opens a font library for use.

Declaration

```
font_lib_t * MGLAPI MGL_openFontLib(  
    const char *libname)
```

Prototype In

mgraph.h

Return Value

Pointer to the opened font library, NULL on error.

Description

Locates the specified font library and opens the file. MGL can load any Windows 2.x style font files (Windows 3.x font files are not supported, but Windows 2.x font files are the standard files even for Windows 3.1. Most resource editors can only output 2.x style font files). Consult the Windows SDK documentation for the format of Windows font files.

If the font library was not found, or an error occurred while reading the font library, this function will return NULL. You can check the *MGL_result* error code to determine the cause.

See Also

MGL_loadFont, *MGL_openFontLibExt*, *MGL_loadFontInstance*, *MGL_unloadFontInstance*, *MGL_closeFontLib*

MGL_openFontLibExt

Load a font file for use from an opened file.

Declaration

```
font_lib_t * MGLAPI MGL_openFontLibExt(
    FILE *f,
    ulong dwOffset,
    ulong dwSize)
```

Prototype In mgraph.h

Parameters

<i>f</i>	Open binary file to read font data from
<i>dwOffset</i>	Offset to start of font in file
<i>dwSize</i>	Size of the file in bytes

Return Value

Pointer to the font data, or NULL on error.

Description

This function is the same as *MGL_loadFont*, however it works with a previously opened file. This allows you to create your own large files with multiple files embedded in them.

See Also

MGL_loadFont, *MGL_openFontLib*, *MGL_unloadFontLib*, *MGL_loadFontInstance*,
MGL_unloadFontInstance

MGL_optimizeRegion

Optimizes the union of rectangles in the region to the minimum number of rectangles.

Declaration

```
void MGLAPI MGL_optimizeRegion(  
    region_t *r)
```

Prototype In

mgraph.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 (*MGL_unionRegion*, *MGL_diffRegion*, *MGL_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

MGL_unionRegion, *MGL_diffRegion*, *MGL_sectRegion*

MGL_packColor

Pack an RGB tuple into an MGL color.

Declaration

```
color_t MGLAPI MGL_packColor(
    pixel_format_t *pf,
    uchar R,
    uchar G,
    uchar B)
```

Prototype In

mgraph.h

Parameters

<i>pf</i>	Pixel format to get packing information from
<i>R</i>	Red color component to pack (0 - 255)
<i>G</i>	Green color component to pack (0 - 255)
<i>B</i>	Blue color component to pack (0 - 255)

Return Value

MGL packed color value appropriate for the specified pixel format information.

Description

This function takes an RGB tuple of 8 bit color components and packs them into an MGL packed color value. The color components are packed according to the specified pixel format information, which can be obtained directly from the mode information for a bitmap or an MGL device context.

See Also

MGL_packColorFast, *MGL_packColorExt*, *MGL_unpackColor*, *MGL_getPixelFormat*

MGL_packColorExt

Pack an RGB with Alpha tuple into an MGL color.

Declaration

```
color_t MGLAPI MGL_packColorExt(
    pixel_format_t *pf,
    uchar A,
    uchar R,
    uchar G,
    uchar B)
```

Prototype In mgraph.h

Parameters

<i>pf</i>	Pixel format to get packing information from
<i>A</i>	Alpha component to pack (0 - 255)
<i>R</i>	Red color component to pack (0 - 255)
<i>G</i>	Green color component to pack (0 - 255)
<i>B</i>	Blue color component to pack (0 - 255)

Return Value

MGL packed color value appropriate for the specified pixel format information.

Description

This function takes an RGB with Alpha tuple of 8 bit color components and packs them into an MGL packed color value. The color components are packed according to the specified pixel format information, which can be obtained directly from the mode information for a bitmap or an MGL device context.

See Also

MGL_packColorFast, *MGL_packColor*, *MGL_unpackColor*, *MGL_getPixelFormat*

MGL_packColorFast

Pack an RGB tuple into an MGL color.

Declaration

```
color_t MGL_packColorFast(  
    pixel_format_t *pf,  
    uchar R,  
    uchar G,  
    uchar B)
```

Prototype In

mgraph.h

Parameters

<i>pf</i>	Pixel format to get packing information from
<i>R</i>	Red color component to pack (0 - 255)
<i>G</i>	Green color component to pack (0 - 255)
<i>B</i>	Blue color component to pack (0 - 255)

Return Value

MGL packed color value appropriate for the specified pixel format information.

Description

This function is the same as *MGL_packColor*, however it is implemented as a macro and hence is more efficient.

See Also

MGL_packColorFastExt, *MGL_packColor*, *MGL_unpackColor*

MGL_packColorFastExt

Pack an RGB with Alpha tuple into an MGL color.

Declaration

```
color_t MGL_packColorFastExt(
    pixel_format_t *pf,
    uchar A,
    uchar R,
    uchar G,
    uchar B)
```

Prototype In

mgraph.h

Parameters

<i>pf</i>	Pixel format to get packing information from
<i>A</i>	Alpha component to pack (0 - 255)
<i>R</i>	Red color component to pack (0 - 255)
<i>G</i>	Green color component to pack (0 - 255)
<i>B</i>	Blue color component to pack (0 - 255)

Return Value

MGL packed color value appropriate for the specified pixel format information.

Description

This function is the same as *MGL_packColorExt*, however it is implemented as a macro and hence is more efficient.

See Also

MGL_packColorFast, *MGL_packColor*, *MGL_unpackColorExt*

MGL_pixel

Draws a pixel at the specified location.

Declaration

```
void MGL_pixel(  
    point_t p)
```

Prototype In

mgraph.h

Parameters

p Point to plot pixel at

Description

This function is the same as *MGL_pixelFast*, however it takes the coordinate of the pixel to plot as a point.

See Also

MGL_pixelCoord

MGL_pixelCoord

Draws a pixel at the specified location.

Declaration

```
void MGLAPI MGL_pixelCoord(  
    int x,  
    int y)
```

Prototype In

mgraph.h

Parameters

<i>x</i>	x coordinate to plot pixel at
<i>y</i>	y coordinate to plot pixel at

Description

Plots a single pixel at the specified location in the current foreground color.

See Also

MGL_pixel

MGL_pixelCoordFast

Draws a pixel at the specified location as fast as possible.

Declaration

```
void MGLAPI MGL_pixelCoordFast(  
    int x,  
    int y)
```

Prototype In

mgraph.h

Parameters

<i>x</i>	x coordinate to plot pixel at
<i>y</i>	y coordinate to plot pixel at

Description

Plots a single pixel at the specified location in the current foreground color. This routine is designed to allow plotting of multiple pixels as fast as possible.

Note that you must call *MGL_beginPixel* before calling this function, and you must call *MGL_endPixel* after you have finished plotting a number of pixels.

See Also

MGL_pixelFast, *MGL_beginPixel*, *MGL_endPixel*.

MGL_pixelFast

Draws a pixel at the specified location as fast as possible.

Declaration

```
void MGL_pixelFast(  
    point_t p)
```

Prototype In

mgraph.h

Parameters

p Point to plot pixel at

Description

This function is the same as *MGL_pixelCoordFast*, however it takes the coordinate of the pixel to plot as a point.

See Also

MGL_pixelCoordFast, *MGL_beginPixel*, *MGL_endPixel*.

MGL_polyLine

Draws a set of connected lines.

Declaration

```
void MGLAPI MGL_polyLine(  
    int count,  
    point_t *vArray)
```

Prototype In

mgraph.h

Parameters

<i>count</i>	Number of vertices in polyline
<i>vArray</i>	Array of vertices in the polyline

Description

This function draws a set of connected line (a polyline). The coordinates of the polyline are specified by *vArray*, and the lines are drawn in the current drawing attributes.

Note that the polyline is not closed by default, so if you wish to draw the outline of a polygon, you will need to add the starting point to the end of the vertex array.

See Also

MGL_polyMarker, *MGL_polyPoint*

MGL_polyPoint

Draws a set of pixels.

Declaration

```
void MGLAPI MGL_polyPoint(  
    int count,  
    point_t *vArray)
```

Prototype In

mgraph.h

Parameters

<i>count</i>	Number of vertices in polyline
<i>vArray</i>	Array of coordinates to draw the pixels at

Description

This function draws a set of pixels in the current color at the locations passed in the vArray parameter.

See Also

MGL_polyLine, *MGL_polyMarker*

MGL_ptInRect

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

Declaration

```
ibool MGL_ptInRect(  
    point_t p,  
    rect_t r)
```

Prototype In

mgraph.h

Parameters

<i>p</i>	Point 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. This version takes the coordinates of the rectangle as two points.

See Also

MGL_ptInRect

MGL_ptInRectCoord

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

Declaration

```
ibool MGL_ptInRectCoord(  
    int x,  
    int y,  
    rect_t r)
```

Prototype In

mgraph.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.

See Also

MGL_ptInRect

MGL_ptInRegion

Determines if a point is contained in a specified region.

Declaration

```
ibool MGL_ptInRegion(  
    point_t p,  
    region_t rgn)
```

Prototype In

mgraph.h

Parameters

<i>p</i>	point structure containing coordinate to test
<i>rgn</i>	Region to test

Return Value

True if the point is contained in the region, false if not.

Description

This function is the same as *MGL_ptInRegionCoord*, however it takes the coordinate of the point to test as a point not two coordinates.

See Also

MGL_ptInRegion

MGL_ptInRegionCoord

Determines if a point is contained in a specified region.

Declaration

```
ibool MGLAPI MGL_ptInRegionCoord(  
    int x,  
    int y,  
    const region_t *rgn)
```

Prototype In

mgraph.h

Parameters

<i>x</i>	x coordinate to test for inclusion
<i>y</i>	y coordinate to test for inclusion
<i>rgn</i>	Region to test

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.

See Also

MGL_ptInRegion

MGL_putBitmap

Draw a lightweight bitmap at the specified location.

Declaration

```
void MGLAPI MGL_putBitmap(
    MGLDC *dc,
    int x,
    int y,
    const bitmap_t *bitmap,
    int op)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to display bitmap on
<i>x</i>	x coordinate to draw bitmap at
<i>y</i>	y coordinate to draw bitmap at
<i>bitmap</i>	Bitmap to display
<i>op</i>	Write mode to use when drawing bitmap

Description

Draws a lightweight bitmap at the specified location. The bitmap can be in any color format, and will be translated as necessary to the color format required by the current device context.

When this function is called for 4 and 8 bit source bitmaps being copied to either 4 or 8bpp destination device contexts, MGL first checks if the color palettes for the source and destination bitmaps are the same. If they are not, MGL translates the pixel values from the source bitmap to the destination color palette, looking for the closest match color if an exact match is not found. In order to obtain maximum performance for blt'ing bitmaps in color index modes, you should ensure that the color palette in the source device matches the color palette in the destination device to avoid on the fly palette translation. If you know in advance that the palette is identical for a series of blit operations, you can turn off all identity palette checking in MGL with the *MGL_checkIdentityPalette* function.

When this function is called for 4 and 8 bit source bitmaps being copied to RGB destination device contexts, MGL will convert the pixels in the source bitmap using the source bitmap palette to map them to the destination pixel format. If however you know in advance that the palette for all source bitmaps is identical for a series of blit operations, you can use the *MGL_checkIdentityPalette* function to disable source palette translation. In this case the MGL will translate all color index bitmaps using the pre-defined color translation palette stored in the destination device context. You would then set the destination device context palette to the common palette for all blit operations using *MGL_setPalette*. If you are translating a lot of color index bitmaps, this

will increase performance by avoiding the need to convert the palette entries to the destination pixel format for every blit operation.

Note also that if the source bitmap palette pointer is set to NULL, palette translation is automatically avoided (ie: has the effect of forcing *MGL_checkIdentityPalette* to false just for that bitmap).

Supported write modes are enumerated in *MGL_writeModeType*.

See Also

MGL_loadBitmap, *MGL_putBitmap*, *MGL_putBitmapSection*, *MGL_putBitmapSrcTrans*,
MGL_putBitmapSrcTransSection, *MGL_putBitmapDstTrans*,
MGL_putBitmapDstTransSection, *MGL_putBitmapMask*, *MGL_putBitmapPatt*,
MGL_putBitmapPattSection, *MGL_putBitmapFx*, *MGL_putBitmapFxSection*,
MGL_stretchBitmap, *MGL_stretchBitmapSection*, *MGL_stretchBitmapFx*,
MGL_stretchBitmapFxSection, *MGL_putIcon*

MGL_putBitmapDstTrans

Draw a transparent lightweight bitmap at the specified location with destination transparency.

Declaration

```
void MGLAPI MGL_putBitmapDstTrans (
    MGLDC *dc,
    int x,
    int y,
    const bitmap_t *bitmap,
    color_t transparent,
    int op)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to display bitmap on
<i>x</i>	x coordinate to draw bitmap at
<i>y</i>	y coordinate to draw bitmap at
<i>bitmap</i>	Bitmap to display
<i>transparent</i>	Transparent color for the bitmap
<i>op</i>	Write mode to use when drawing bitmap

Description

Draws a transparent lightweight bitmap at the specified location with destination transparency. When transferring the data with destination transparency, pixels in the destination image that are equal to the specified transparent color will be updated, and those pixels that are not the same will be skipped. This is effectively the operation performed for 'blueScreen'ing or color keying and can also be used for drawing transparent sprites. Note however that destination transparency is very slow in software compared to source transparency!

Note: *If you are doing pixel format conversion at the same time (ie: color depth for source bitmap is different to the destination bitmap), then the transparent color value must be set to the traslated destination pixel format. Ie: if you are blitting an 8bpp bitmap to a 32bpp device context, the transparent color must be a 32bpp value.*

Note: *This routine also only works with pixel depths that are at least 4 bits deep.*

See Also

MGL_loadBitmap, MGL_putBitmap, MGL_putBitmapSection, MGL_putBitmapSrcTrans, MGL_putBitmapSrcTransSection, MGL_putBitmapDstTrans, MGL_putBitmapDstTransSection, MGL_putBitmapMask, MGL_putBitmapPatt, MGL_putBitmapPattSection, MGL_putBitmapFx, MGL_putBitmapFxSection, MGL_stretchBitmap, MGL_stretchBitmapSection, MGL_stretchBitmapFx, MGL_stretchBitmapFxSection, MGL_putIcon

MGL_putBitmapDstTransSection

Draw a section of a transparent lightweight bitmap at the specified location with destination transparency.

Declaration

```
void MGLAPI MGL_putBitmapDstTransSection(
    MGLDC *dc,
    int left,
    int top,
    int right,
    int bottom,
    int dstLeft,
    int dstTop,
    const bitmap_t *bitmap,
    color_t transparent,
    int op)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to display bitmap on
<i>left</i>	Left coordinate of section to draw
<i>top</i>	Top coordinate of section to draw
<i>right</i>	Right coordinate of section to draw
<i>bottom</i>	Bottom coordinate of section to draw
<i>dstLeft</i>	Left coordinate of destination of bitmap section
<i>dstTop</i>	Right coordinate for destination of bitmap section
<i>bitmap</i>	Bitmap to display
<i>transparent</i>	Transparent color for the bitmap
<i>op</i>	Write mode to use when drawing bitmap

Description

Draws a section of a transparent lightweight bitmap at the specified location with destination transparency. When transferring the data with destination transparency, pixels in the destination image that are equal to the specified transparent color will be updated, and those pixels that are not the same will be skipped. This is effectively the operation performed for 'blueScreen'ing or color keying and can also be used for drawing transparent sprites. Note however that destination transparency is very slow in software compared to source transparency!

Note: *If you are doing pixel format conversion at the same time (ie: color depth for source bitmap is different to the destination bitmap), then the transparent color value must be set to the traslated destination pixel format. Ie: if you are blitting an 8bpp bitmap to a 32bpp device context, the transparent color must be a 32bpp value.*

Note: *This routine also only works with pixel depths that are at least 4 bits deep.*

See Also

MGL_loadBitmap, MGL_putBitmap, MGL_putBitmapSection, MGL_putBitmapSrcTrans, MGL_putBitmapSrcTransSection, MGL_putBitmapDstTrans, MGL_putBitmapDstTransSection, MGL_putBitmapMask, MGL_putBitmapPatt, MGL_putBitmapPattSection, MGL_putBitmapFx, MGL_putBitmapFxSection, MGL_stretchBitmap, MGL_stretchBitmapSection, MGL_stretchBitmapFx, MGL_stretchBitmapFxSection, MGL_putIcon

MGL_putBitmapFx

Draw a lightweight bitmap at the specified location, while applying different effects in the process.

Declaration

```
void MGLAPI MGL_putBitmapFx(
    MGLDC *dc,
    int x,
    int y,
    const bitmap_t *bitmap,
    bltfx_t *fx)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to display bitmap on
<i>x</i>	x coordinate to draw bitmap at
<i>y</i>	y coordinate to draw bitmap at
<i>bitmap</i>	Bitmap to display
<i>fx</i>	Information describing the effects to apply to the blit (<i>bltfx_t</i>)

Description

Draws a lightweight bitmap at the specified location, with optional effects applied. The effects applied to the blit operation range from X and Y bitmap flipping to color transparency and blending. All of these operations can be applied to data being copied between different device context of different color depths and pixel formats if desired (ie: 32-bit ARGB alpha blended bitmap to a 16-bit device context). Please refer to the documentation for the *bltfx_t* structure and the *MGL_bitBltFxFlagsType* enumeration that defines the flags passed to this function.

When this function is called for 4 and 8 bit source bitmaps being copied to either 4 or 8bpp destination device contexts, MGL first checks if the color palettes for the source and destination bitmaps are the same. If they are not, MGL translates the pixel values from the source bitmap to the destination color palette, looking for the closest match color if an exact match is not found. In order to obtain maximum performance for blt'ing bitmaps in color index modes, you should ensure that the color palette in the source device matches the color palette in the destination device to avoid on the fly palette translation. If you know in advance that the palette is identical for a series of blit operations, you can turn off all identity palette checking in MGL with the *MGL_checkIdentityPalette* function.

When this function is called for 4 and 8 bit source bitmaps being copied to RGB destination device contexts, MGL will convert the pixels in the source bitmap using the source bitmap palette to map them to the destination pixel format. If however you know in advance that the palette for all source bitmaps is identical for a series of blit operations, you can use the *MGL_checkIdentityPalette* function to disable source palette

translation. In this case the MGL will translate all color index bitmaps using the pre-defined color translation palette stored in the destination device context. You would then set the destination device context palette to the common palette for all blit operations using *MGL_setPalette*. If you are translating a lot of color index bitmaps, this will increase performance by avoiding the need to convert the palette entries to the destination pixel format for every blit operation.

Note also that if the source bitmap palette pointer is set to NULL, palette translation is automatically avoided (ie: has the effect of forcing *MGL_checkIdentityPalette* to false just for that bitmap).

See Also

MGL_loadBitmap, *MGL_putBitmap*, *MGL_putBitmapSection*, *MGL_putBitmapSrcTrans*,
MGL_putBitmapSrcTransSection, *MGL_putBitmapDstTrans*,
MGL_putBitmapDstTransSection, *MGL_putBitmapMask*, *MGL_putBitmapPatt*,
MGL_putBitmapPattSection, *MGL_putBitmapFx*, *MGL_putBitmapFxSection*,
MGL_stretchBitmap, *MGL_stretchBitmapSection*, *MGL_stretchBitmapFx*,
MGL_stretchBitmapFxSection, *MGL_putIcon*

MGL_putBitmapFxSection

Draw a section of a lightweight bitmap at the specified location, while applying different effects in the process.

Declaration

```
void MGLAPI MGL_putBitmapFxSection(
    MGLDC *dc,
    int left,
    int top,
    int right,
    int bottom,
    int dstLeft,
    int dstTop,
    const bitmap_t *bitmap,
    bltfx_t *fx)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to display bitmap on
<i>left</i>	Left coordinate of section to draw
<i>top</i>	Top coordinate of section to draw
<i>right</i>	Right coordinate of section to draw
<i>bottom</i>	Bottom coordinate of section to draw
<i>dstLeft</i>	Left coordinate of destination of bitmap section
<i>dstTop</i>	Right coordinate for destination of bitmap section
<i>bitmap</i>	Bitmap to display
<i>fx</i>	Information describing the effects to apply to the blit (<i>bltfx_t</i>)

Description

Draws a section of a lightweight bitmap at the specified location, with optional effects applied. The effects applied to the blit operation range from X and Y bitmap flipping to color transparency and blending. All of these operations can be applied to data being copied between different device context of different color depths and pixel formats if desired (ie: 32-bit ARGB alpha blended bitmap to a 16-bit device context). Please refer to the documentation for the *bltfx_t* structure and the *MGL_bitBltFxFlagsType* enumeration that defines the flags passed to this function.

When this function is called for 4 and 8 bit source bitmaps being copied to either 4 or 8bpp destination device contexts, MGL first checks if the color palettes for the source and destination bitmaps are the same. If they are not, MGL translates the pixel values from the source bitmap to the destination color palette, looking for the closest match color if an exact match is not found. In order to obtain maximum performance for blt'ing bitmaps in color index modes, you should ensure that the color palette in the source device matches the color palette in the destination device to avoid on the fly palette translation. If you know in advance that the palette is identical for a series of blit

operations, you can turn off all identity palette checking in MGL with the *MGL_checkIdentityPalette* function.

When this function is called for 4 and 8 bit source bitmaps being copied to RGB destination device contexts, MGL will convert the pixels in the source bitmap using the source bitmap palette to map them to the destination pixel format. If however you know in advance that the palette for all source bitmaps is identical for a series of blit operations, you can use the *MGL_checkIdentityPalette* function to disable source palette translation. In this case the MGL will translate all color index bitmaps using the pre-defined color translation palette stored in the destination device context. You would then set the destination device context palette to the common palette for all blit operations using *MGL_setPalette*. If you are translating a lot of color index bitmaps, this will increase performance by avoiding the need to convert the palette entries to the destination pixel format for every blit operation.

Note also that if the source bitmap palette pointer is set to NULL, palette translation is automatically avoided (ie: has the effect of forcing *MGL_checkIdentityPalette* to false just for that bitmap).

See Also

MGL_loadBitmap, *MGL_putBitmap*, *MGL_putBitmapSection*, *MGL_putBitmapSrcTrans*,
MGL_putBitmapSrcTransSection, *MGL_putBitmapDstTrans*,
MGL_putBitmapDstTransSection, *MGL_putBitmapMask*, *MGL_putBitmapPatt*,
MGL_putBitmapPattSection, *MGL_putBitmapFx*, *MGL_putBitmapFxSection*,
MGL_stretchBitmap, *MGL_stretchBitmapSection*, *MGL_stretchBitmapFx*,
MGL_stretchBitmapFxSection, *MGL_putIcon*

MGL_putBitmapMask

Draw a lightweight bitmap mask in the specified color.

Declaration

```
void MGLAPI MGL_putBitmapMask(
    MGLDC *dc,
    int x,
    int y,
    const bitmap_t *mask,
    color_t color,
    int op)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to display bitmap on
<i>x</i>	x coordinate to draw bitmap at
<i>y</i>	y coordinate to draw bitmap at
<i>mask</i>	Monochrome bitmap mask to display
<i>color</i>	Color to draw in
<i>op</i>	Write mode to use when drawing bitmap

Description

Draws a lightweight monochrome bitmap at the specified location. This is just a simply utility function that draws the monochrome bitmap in a specified color and write mode as fast as possible.

See Also

MGL_loadBitmap, MGL_putBitmap, MGL_putBitmapSection, MGL_putBitmapSrcTrans, MGL_putBitmapSrcTransSection, MGL_putBitmapDstTrans, MGL_putBitmapDstTransSection, MGL_putBitmapMask, MGL_putBitmapPatt, MGL_putBitmapPattSection, MGL_putBitmapFx, MGL_putBitmapFxSection, MGL_stretchBitmap, MGL_stretchBitmapSection, MGL_stretchBitmapFx, MGL_stretchBitmapFxSection, MGL_putIcon

MGL_putBitmapPatt

Draw a lightweight bitmap at the specified location while applying a mono or color pattern.

Declaration

```
void MGLAPI MGL_putBitmapPatt(
    MGLDC *dc,
    int x,
    int y,
    const bitmap_t *bitmap,
    int usePixmap,
    int rop3)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to display bitmap on
<i>x</i>	x coordinate to draw bitmap at
<i>y</i>	y coordinate to draw bitmap at
<i>bitmap</i>	Bitmap to display
<i>usePixmap</i>	True to use color pixmap pattern, false for mono bitmap pattern
<i>rop3</i>	ROP3 raster operation code to use during Blt (<i>MGL_rop3CodesType</i>)

Description

Draws a lightweight bitmap at the specified location, while applying either a mono bitmap pattern or a color pixmap pattern to the data with a ternary raster operation code (ROP3). If the usePixmap parameter is set to true, the current pixmap pattern set by *MGL_setPenPixmapPattern* will be applied as pattern data, otherwise the current monochrome bitmap pattern set by *MGL_setPenBitmapPattern* will be applied.

See Also

MGL_loadBitmap, *MGL_putBitmap*, *MGL_putBitmapSection*, *MGL_putBitmapSrcTrans*, *MGL_putBitmapSrcTransSection*, *MGL_putBitmapDstTrans*, *MGL_putBitmapDstTransSection*, *MGL_putBitmapMask*, *MGL_putBitmapPatt*, *MGL_putBitmapPattSection*, *MGL_putBitmapFx*, *MGL_putBitmapFxSection*, *MGL_stretchBitmap*, *MGL_stretchBitmapSection*, *MGL_stretchBitmapFx*, *MGL_stretchBitmapFxSection*, *MGL_putIcon*

MGL_putBitmapPattSection

Draw a section of a lightweight bitmap at the specified location while applying a mono or color pattern.

Declaration

```
void MGLAPI MGL_putBitmapPattSection(
    MGLDC *dc,
    int left,
    int top,
    int right,
    int bottom,
    int dstLeft,
    int dstTop,
    const bitmap_t *bitmap,
    int usePixMap,
    int rop3)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to display bitmap on
<i>left</i>	Left coordinate of section to draw
<i>top</i>	Top coordinate of section to draw
<i>right</i>	Right coordinate of section to draw
<i>bottom</i>	Bottom coordinate of section to draw
<i>dstLeft</i>	Left coordinate of destination of bitmap section
<i>dstTop</i>	Right coordinate for destination of bitmap section
<i>bitmap</i>	Bitmap to display
<i>usePixMap</i>	True to use color pixmap pattern, false for mono bitmap pattern
<i>rop3</i>	ROP3 raster operation code to use during Blt (<i>MGL_rop3CodesType</i>)

Description

Draws a section of lightweight bitmap at the specified location, while applying either a mono bitmap pattern or a color pixmap pattern to the data with a ternary raster operation code (ROP3). If the usePixMap parameter is set to true, the current pixmap pattern set by *MGL_setPenPixmapPattern* will be applied as pattern data, otherwise the current monochrome bitmap pattern set by *MGL_setPenBitmapPattern* will be applied.

See Also

MGL_loadBitmap, *MGL_putBitmap*, *MGL_putBitmapSection*, *MGL_putBitmapSrcTrans*, *MGL_putBitmapSrcTransSection*, *MGL_putBitmapDstTrans*, *MGL_putBitmapDstTransSection*, *MGL_putBitmapMask*, *MGL_putBitmapPatt*, *MGL_putBitmapPattSection*, *MGL_putBitmapFx*, *MGL_putBitmapFxSection*,

*MGL_stretchBitmap, MGL_stretchBitmapSection, MGL_stretchBitmapEx,
MGL_stretchBitmapExSection, MGL_putIcon*

MGL_putBitmapSection

Draw a section of a lightweight bitmap at the specified location.

Declaration

```
void MGLAPI MGL_putBitmapSection(
    MGLDC *dc,
    int left,
    int top,
    int right,
    int bottom,
    int dstLeft,
    int dstTop,
    const bitmap_t *bitmap,
    int op)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to display bitmap on
<i>left</i>	Left coordinate of section to draw
<i>top</i>	Top coordinate of section to draw
<i>right</i>	Right coordinate of section to draw
<i>bottom</i>	Bottom coordinate of section to draw
<i>dstLeft</i>	Left coordinate of destination of bitmap section
<i>dstTop</i>	Right coordinate for destination of bitmap section
<i>bitmap</i>	Bitmap to display
<i>op</i>	Write mode to use when drawing bitmap

Description

Draws a section of a lightweight bitmap at the specified location. The bitmap can be in any color format, and will be translated as necessary to the color format required by the current device context.

When this function is called for 4 and 8 bit source bitmaps being copied to either 4 or 8bpp destination device contexts, MGL first checks if the color palettes for the source and destination bitmaps are the same. If they are not, MGL translates the pixel values from the source bitmap to the destination color palette, looking for the closest match color if an exact match is not found. In order to obtain maximum performance for blt'ing bitmaps in color index modes, you should ensure that the color palette in the source device matches the color palette in the destination device to avoid on the fly palette translation. If you know in advance that the palette is identical for a series of blit operations, you can turn off all identity palette checking in MGL with the *MGL_checkIdentityPalette* function.

When this function is called for 4 and 8 bit source bitmaps being copied to RGB destination device contexts, MGL will convert the pixels in the source bitmap using the

source bitmap palette to map them to the destination pixel format. If however you know in advance that the palette for all source bitmaps is identical for a series of blit operations, you can use the *MGL_checkIdentityPalette* function to disable source palette translation. In this case the MGL will translate all color index bitmaps using the pre-defined color translation palette stored in the destination device context. You would then set the destination device context palette to the common palette for all blit operations using *MGL_setPalette*. If you are translating a lot of color index bitmaps, this will increase performance by avoiding the need to convert the palette entries to the destination pixel format for every blit operation.

Note also that if the source bitmap palette pointer is set to NULL, palette translation is automatically avoided (ie: has the effect of forcing *MGL_checkIdentityPalette* to false just for that bitmap).

Supported write modes are enumerated in *MGL_writeModeType*.

See Also

MGL_loadBitmap, *MGL_putBitmap*, *MGL_putBitmapSection*, *MGL_putBitmapSrcTrans*,
MGL_putBitmapSrcTransSection, *MGL_putBitmapDstTrans*,
MGL_putBitmapDstTransSection, *MGL_putBitmapMask*, *MGL_putBitmapPatt*,
MGL_putBitmapPattSection, *MGL_putBitmapFx*, *MGL_putBitmapFxSection*,
MGL_stretchBitmap, *MGL_stretchBitmapSection*, *MGL_stretchBitmapFx*,
MGL_stretchBitmapFxSection, *MGL_putIcon*

MGL_putBitmapSrcTrans

Draw a transparent lightweight bitmap at the specified location with source transparency.

Declaration

```
void MGLAPI MGL_putBitmapSrcTrans(
    MGLDC *dc,
    int x,
    int y,
    const bitmap_t *bitmap,
    color_t transparent,
    int op)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to display bitmap on
<i>x</i>	x coordinate to draw bitmap at
<i>y</i>	y coordinate to draw bitmap at
<i>bitmap</i>	Bitmap to display
<i>transparent</i>	Transparent color for the bitmap
<i>op</i>	Write mode to use when drawing bitmap

Description

Draws a transparent lightweight bitmap at the specified location with source transparency. When transferring the data with source transparency, for pixels in the source image that are equal to the specified transparent color, the related pixel in the destination buffer will remain untouched. This allows you to quickly transfer sprites between device contexts with a single color being allocated as a transparent color.

Note: *If you are doing pixel format conversion at the same time (ie: color depth for source bitmap is different to the destination bitmap), then the transparent color value must be set to the traslated destination pixel format. Ie: if you are blitting an 8bpp bitmap to a 32bpp device context, the transparent color must be a 32bpp value.*

Note: *This routine also only works with pixel depths that are at least 4 bits deep.*

See Also

MGL_loadBitmap, MGL_putBitmap, MGL_putBitmapSection, MGL_putBitmapSrcTrans, MGL_putBitmapSrcTransSection, MGL_putBitmapDstTrans, MGL_putBitmapDstTransSection, MGL_putBitmapMask, MGL_putBitmapPatt, MGL_putBitmapPattSection, MGL_putBitmapFx, MGL_putBitmapFxSection, MGL_stretchBitmap, MGL_stretchBitmapSection, MGL_stretchBitmapFx, MGL_stretchBitmapFxSection, MGL_putIcon

MGL_putBitmapSrcTransSection

Draw a section of a transparent lightweight bitmap at the specified location with source transparency.

Declaration

```
void MGLAPI MGL_putBitmapSrcTransSection(
    MGLDC *dc,
    int left,
    int top,
    int right,
    int bottom,
    int dstLeft,
    int dstTop,
    const bitmap_t *bitmap,
    color_t transparent,
    int op)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to display bitmap on
<i>left</i>	Left coordinate of section to draw
<i>top</i>	Top coordinate of section to draw
<i>right</i>	Right coordinate of section to draw
<i>bottom</i>	Bottom coordinate of section to draw
<i>dstLeft</i>	Left coordinate of destination of bitmap section
<i>dstTop</i>	Right coordinate for destination of bitmap section
<i>bitmap</i>	Bitmap to display
<i>transparent</i>	Transparent color for the bitmap
<i>op</i>	Write mode to use when drawing bitmap

Description

Draws a section of a transparent lightweight bitmap at the specified location with source transparency. When transferring the data with source transparency, pixels in the source image that are equal to the specified transparent color, the related pixel in the destination buffer will remain untouched. This allows you to quickly transfer sprites between device contexts with a single color being allocated as a transparent color.

Note: *If you are doing pixel format conversion at the same time (ie: color depth for source bitmap is different to the destination bitmap), then the transparent color value must be set to the traslated destination pixel format. Ie: if you are blitting an 8bpp bitmap to a 32bpp device context, the transparent color must be a 32bpp value.*

Note: *This routine also only works with pixel depths that are at least 4 bits deep.*

See Also

*MGL_loadBitmap, MGL_putBitmap, MGL_putBitmapSection, MGL_putBitmapSrcTrans,
MGL_putBitmapSrcTransSection, MGL_putBitmapDstTrans,
MGL_putBitmapDstTransSection, MGL_putBitmapMask, MGL_putBitmapPatt,
MGL_putBitmapPattSection, MGL_putBitmapFx, MGL_putBitmapFxSection,
MGL_stretchBitmap, MGL_stretchBitmapSection, MGL_stretchBitmapFx,
MGL_stretchBitmapFxSection, MGL_putIcon*

MGL_putBuffer

Copy an offscreen buffer to the specified location.

Declaration

```
void MGLAPI MGL_putBuffer(
    MGLDC *dc,
    int x,
    int y,
    MGLBUF *buf,
    int op)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to copy buffer to
<i>x</i>	x coordinate to copy buffer to
<i>y</i>	y coordinate to copy buffer to
<i>buf</i>	Buffer to copy
<i>op</i>	Write mode to use when drawing buffer

Description

Copies an offscreen buffer to the specified location on the device context. Supported write modes are enumerated in *MGL_writeModeType*.

Note: *This function will fail if you attempt to copy a buffer to a device context that is not the device context that the original buffer was allocated for.*

See Also

MGL_copyToBuffer, MGL_copyBitmapToBuffer, MGL_updateBufferCache, MGL_updateFromBufferCache, MGL_putBuffer, MGL_putBufferSection, MGL_putBufferSrcTrans, MGL_putBufferSrcTransSection, MGL_putBufferDstTrans, MGL_putBufferDstTransSection, MGL_putBufferPatt, MGL_putBufferPattSection, MGL_putBufferFx, MGL_putBufferFxSection, MGL_stretchBuffer, MGL_stretchBufferSection, MGL_stretchBufferFx, MGL_stretchBufferFxSection

MGL_putBufferDstTrans

Copy an offscreen buffer to the specified location with destination transparency.

Declaration

```
void MGLAPI MGL_putBufferDstTrans (
    MGLDC *dc,
    int x,
    int y,
    MGLBUF *buf,
    color_t transparent,
    int op)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to copy buffer to
<i>x</i>	x coordinate to copy buffer to
<i>y</i>	y coordinate to copy buffer to
<i>buf</i>	Buffer to copy
<i>transparent</i>	Transparent color for the bitmap
<i>op</i>	Write mode to use when drawing buffer

Description

Copies a offscreen buffer to the specified location on the device context with either source or destination transparency.

When transferring the data with destination transparency, pixels in the destination image that are equal to the specified transparent color will be updated, and those pixels that are not the same will be skipped. This is effectively the operation performed for 'blueScreen'ing or color keying and can also be used for drawing transparent sprites. Note however that destination transparency is very slow in software compared to source transparency!

Note: *This function will fail if you attempt to copy a buffer to a device context that is not the device context that the original buffer was allocated for.*

See Also

MGL_copyToBuffer, MGL_copyBitmapToBuffer, MGL_updateBufferCache, MGL_updateFromBufferCache, MGL_putBuffer, MGL_putBufferSection, MGL_putBufferSrcTrans, MGL_putBufferSrcTransSection, MGL_putBufferDstTrans, MGL_putBufferDstTransSection, MGL_putBufferPatt, MGL_putBufferPattSection, MGL_putBufferFx, MGL_putBufferFxSection, MGL_stretchBuffer, MGL_stretchBufferSection, MGL_stretchBufferFx, MGL_stretchBufferFxSection

MGL_putBufferDstTransSection

Copy a section of an offscreen buffer to the specified location with destination transparency.

Declaration

```
void MGLAPI MGL_putBufferDstTransSection(
    MGLDC *dc,
    int left,
    int top,
    int right,
    int bottom,
    int dstLeft,
    int dstTop,
    MGLBUF *buf,
    color_t transparent,
    int op)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to copy buffer to
<i>left</i>	Left coordinate of section to copy
<i>top</i>	Top coordinate of section to copy
<i>right</i>	Right coordinate of section to copy
<i>bottom</i>	Bottom coordinate of section to copy
<i>dstLeft</i>	Left coordinate of destination to copy buffer to
<i>dstTop</i>	Right coordinate of destination to copy buffer to
<i>buf</i>	Buffer to copy
<i>transparent</i>	Transparent color for the bitmap
<i>op</i>	Write mode to use when drawing buffer

Description

Copies a section of an offscreen buffer to the specified location on the device context with either source or destination transparency.

When transferring the data with destination transparency, pixels in the destination image that are equal to the specified transparent color will be updated, and those pixels that are not the same will be skipped. This is effectively the operation performed for 'blueScreen'ing or color keying and can also be used for drawing transparent sprites. Note however that destination transparency is very slow in software compared to source transparency!

Note: *This function will fail if you attempt to copy a buffer to a device context that is not the device context that the original buffer was allocated for.*

See Also

*MGL_copyToBuffer, MGL_copyBitmapToBuffer, MGL_updateBufferCache,
MGL_updateFromBufferCache, MGL_putBuffer, MGL_putBufferSection,
MGL_putBufferSrcTrans, MGL_putBufferSrcTransSection, MGL_putBufferDstTrans,
MGL_putBufferDstTransSection, MGL_putBufferPatt, MGL_putBufferPattSection,
MGL_putBufferFx, MGL_putBufferFxSection, MGL_stretchBuffer, MGL_stretchBufferSection,
MGL_stretchBufferFx, MGL_stretchBufferFxSection*

MGL_putBufferFx

Copy an offscreen buffer to the specified location, while applying different effects in the process.

Declaration

```
void MGLAPI MGL_putBufferFx(
    MGLDC *dc,
    int x,
    int y,
    MGLBUF *buf,
    bltfx_t *fx)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to copy buffer to
<i>x</i>	x coordinate to copy buffer to
<i>y</i>	y coordinate to copy buffer to
<i>buf</i>	Buffer to copy
<i>fx</i>	Information describing the effects to apply to the blit (<i>bltfx_t</i>)

Description

Copies an offscreen buffer to the specified location on the device context, with optional effects applied. The effects applied to the blit operation range from X and Y bitmap flipping to color transparency and blending.

Note: *This function will fail if you attempt to copy a buffer to a device context that is not the device context that the original buffer was allocated for.*

See Also

MGL_copyToBuffer, MGL_copyBitmapToBuffer, MGL_updateBufferCache, MGL_updateFromBufferCache, MGL_putBuffer, MGL_putBufferSection, MGL_putBufferSrcTrans, MGL_putBufferSrcTransSection, MGL_putBufferDstTrans, MGL_putBufferDstTransSection, MGL_putBufferPatt, MGL_putBufferPattSection, MGL_putBufferFx, MGL_putBufferFxSection, MGL_stretchBuffer, MGL_stretchBufferSection, MGL_stretchBufferFx, MGL_stretchBufferFxSection

MGL_putBufferFxSection

Copy a section of an offscreen buffer to the specified location, while applying different effects in the process.

Declaration

```
void MGLAPI MGL_putBufferFxSection(
    MGLDC *dc,
    int left,
    int top,
    int right,
    int bottom,
    int dstLeft,
    int dstTop,
    MGLBUF *buf,
    bltfx_t *fx)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to copy buffer to
<i>left</i>	Left coordinate of section to copy
<i>top</i>	Top coordinate of section to copy
<i>right</i>	Right coordinate of section to copy
<i>bottom</i>	Bottom coordinate of section to copy
<i>dstLeft</i>	Left coordinate of destination to copy buffer to
<i>dstTop</i>	Right coordinate of destination to copy buffer to
<i>buf</i>	Buffer to copy
<i>fx</i>	Information describing the effects to apply to the blit (<i>bltfx_t</i>)

Description

Copies a section of an offscreen buffer to the specified location on the device context, with optional effects applied. The effects applied to the blit operation range from X and Y bitmap flipping to color transparency and blending.

Note: *This function will fail if you attempt to copy a buffer to a device context that is not the device context that the original buffer was allocated for.*

See Also

MGL_copyToBuffer, MGL_copyBitmapToBuffer, MGL_updateBufferCache, MGL_updateFromBufferCache, MGL_putBuffer, MGL_putBufferSection, MGL_putBufferSrcTrans, MGL_putBufferSrcTransSection, MGL_putBufferDstTrans, MGL_putBufferDstTransSection, MGL_putBufferPatt, MGL_putBufferPattSection, MGL_putBufferFx, MGL_putBufferFxSection, MGL_stretchBuffer, MGL_stretchBufferSection, MGL_stretchBufferFx, MGL_stretchBufferFxSection

MGL_putBufferPatt

Copy an offscreen buffer to the specified location with an optional pattern.

Declaration

```
void MGLAPI MGL_putBufferPatt(
    MGLDC *dc,
    int x,
    int y,
    MGLBUF *buf,
    int usePixmap,
    int rop3)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to copy buffer to
<i>x</i>	x coordinate to copy buffer to
<i>y</i>	y coordinate to copy buffer to
<i>buf</i>	Buffer to copy
<i>usePixmap</i>	True to use color pixmap pattern, false for mono bitmap pattern
<i>rop3</i>	ROP3 raster operation code to use during Blt (<i>MGL_rop3CodesType</i>)

Description

Copies an offscreen buffer to the specified location on the device context, while applying either a mono bitmap pattern or a color pixmap pattern to the data with a ternary raster operation code (ROP3). If the usePixmap parameter is set to true, the current pixmap pattern set by *MGL_setPenPixmapPattern* will be applied as pattern data, otherwise the current monochrome bitmap pattern set by *MGL_setPenBitmapPattern* will be applied.

Note: *This function will fail if you attempt to copy a buffer to a device context that is not the device context that the original buffer was allocated for.*

See Also

MGL_copyToBuffer, MGL_copyBitmapToBuffer, MGL_updateBufferCache, MGL_updateFromBufferCache, MGL_putBuffer, MGL_putBufferSection, MGL_putBufferSrcTrans, MGL_putBufferSrcTransSection, MGL_putBufferDstTrans, MGL_putBufferDstTransSection, MGL_putBufferPatt, MGL_putBufferPattSection, MGL_putBufferFx, MGL_putBufferFxSection, MGL_stretchBuffer, MGL_stretchBufferSection, MGL_stretchBufferFx, MGL_stretchBufferFxSection

MGL_putBufferPattSection

Copy a section of an offscreen buffer to the specified location with an optional pattern.

Declaration

```
void MGLAPI MGL_putBufferPattSection(
    MGLDC *dc,
    int left,
    int top,
    int right,
    int bottom,
    int dstLeft,
    int dstTop,
    MGLBUF *buf,
    int usePixmap,
    int rop3)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to copy buffer to
<i>left</i>	Left coordinate of section to copy
<i>top</i>	Top coordinate of section to copy
<i>right</i>	Right coordinate of section to copy
<i>bottom</i>	Bottom coordinate of section to copy
<i>dstLeft</i>	Left coordinate of destination to copy buffer to
<i>dstTop</i>	Right coordinate of destination to copy buffer to
<i>buf</i>	Buffer to copy
<i>usePixmap</i>	True to use color pixmap pattern, false for mono bitmap pattern
<i>rop3</i>	ROP3 raster operation code to use during Blt (<i>MGL_rop3CodesType</i>)

Description

Copies a section of an offscreen buffer to the specified location on the device context, while applying either a mono bitmap pattern or a color pixmap pattern to the data with a ternary raster operation code (ROP3). If the *usePixmap* parameter is set to true, the current pixmap pattern set by *MGL_setPenPixmapPattern* will be applied as pattern data, otherwise the current monochrome bitmap pattern set by *MGL_setPenBitmapPattern* will be applied.

Note: *This function will fail if you attempt to copy a buffer to a device context that is not the device context that the original buffer was allocated for.*

See Also

MGL_copyToBuffer, *MGL_copyBitmapToBuffer*, *MGL_updateBufferCache*, *MGL_updateFromBufferCache*, *MGL_putBuffer*, *MGL_putBufferSection*,

*MGL_putBufferSrcTrans, MGL_putBufferSrcTransSection, MGL_putBufferDstTrans,
MGL_putBufferDstTransSection, MGL_putBufferPatt, MGL_putBufferPattSection,
MGL_putBufferFx, MGL_putBufferFxSection, MGL_stretchBuffer, MGL_stretchBufferSection,
MGL_stretchBufferFx, MGL_stretchBufferFxSection*

MGL_putBufferSection

Copy a section of an offscreen buffer to the specified location.

Declaration

```
void MGLAPI MGL_putBufferSection(
    MGLDC *dc,
    int left,
    int top,
    int right,
    int bottom,
    int dstLeft,
    int dstTop,
    MGLBUF *buf,
    int op)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to copy buffer to
<i>left</i>	Left coordinate of section to copy
<i>top</i>	Top coordinate of section to copy
<i>right</i>	Right coordinate of section to copy
<i>bottom</i>	Bottom coordinate of section to copy
<i>dstLeft</i>	Left coordinate of destination to copy buffer to
<i>dstTop</i>	Right coordinate of destination to copy buffer to
<i>buf</i>	Buffer to copy
<i>op</i>	Write mode to use when drawing buffer

Description

Copies a section of an offscreen buffer to the specified location on the device context.

Note: *This function will fail if you attempt to copy a buffer to a device context that is not the device context that the original buffer was allocated for.*

See Also

MGL_copyToBuffer, MGL_copyBitmapToBuffer, MGL_updateBufferCache, MGL_updateFromBufferCache, MGL_putBuffer, MGL_putBufferSection, MGL_putBufferSrcTrans, MGL_putBufferSrcTransSection, MGL_putBufferDstTrans, MGL_putBufferDstTransSection, MGL_putBufferPatt, MGL_putBufferPattSection, MGL_putBufferFx, MGL_putBufferFxSection, MGL_stretchBuffer, MGL_stretchBufferSection, MGL_stretchBufferFx, MGL_stretchBufferFxSection

MGL_putBufferSrcTrans

Copy an offscreen buffer to the specified location with source transparency.

Declaration

```
void MGLAPI MGL_putBufferSrcTrans (
    MGLDC *dc,
    int x,
    int y,
    MGLBUF *buf,
    color_t transparent,
    int op)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to copy buffer to
<i>x</i>	x coordinate to copy buffer to
<i>y</i>	y coordinate to copy buffer to
<i>buf</i>	Buffer to copy
<i>transparent</i>	Transparent color for the bitmap
<i>op</i>	Write mode to use when drawing buffer

Description

Copies a offscreen buffer to the specified location on the device context with either source or destination transparency.

When transferring the data with source transparency, for pixels in the source image that are equal to the specified transparent color, the related pixel in the destination buffer will remain untouched. This allows you to quickly transfer sprites between device contexts with a single color being allocated as a transparent color.

Note: *This function will fail if you attempt to copy a buffer to a device context that is not the device context that the original buffer was allocated for.*

See Also

MGL_copyToBuffer, MGL_copyBitmapToBuffer, MGL_updateBufferCache, MGL_updateFromBufferCache, MGL_putBuffer, MGL_putBufferSection, MGL_putBufferSrcTrans, MGL_putBufferSrcTransSection, MGL_putBufferDstTrans, MGL_putBufferDstTransSection, MGL_putBufferPatt, MGL_putBufferPattSection, MGL_putBufferFx, MGL_putBufferFxSection, MGL_stretchBuffer, MGL_stretchBufferSection, MGL_stretchBufferFx, MGL_stretchBufferFxSection

MGL_putBufferSrcTransSection

Copy a section of an offscreen buffer to the specified location with source transparency.

Declaration

```
void MGLAPI MGL_putBufferSrcTransSection(
    MGLDC *dc,
    int left,
    int top,
    int right,
    int bottom,
    int dstLeft,
    int dstTop,
    MGLBUF *buf,
    color_t transparent,
    int op)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to copy buffer to
<i>left</i>	Left coordinate of section to copy
<i>top</i>	Top coordinate of section to copy
<i>right</i>	Right coordinate of section to copy
<i>bottom</i>	Bottom coordinate of section to copy
<i>dstLeft</i>	Left coordinate of destination to copy buffer to
<i>dstTop</i>	Right coordinate of destination to copy buffer to
<i>buf</i>	Buffer to copy
<i>transparent</i>	Transparent color for the bitmap
<i>op</i>	Write mode to use when drawing buffer

Description

Copies a section of an offscreen buffer to the specified location on the device context with either source or destination transparency.

When transferring the data with source transparency, pixels in the source image that are equal to the specified transparent color, the related pixel in the destination buffer will remain untouched. This allows you to quickly transfer sprites between device contexts with a single color being allocated as a transparent color.

Note: This function will fail if you attempt to copy a buffer to a device context that is not the device context that the original buffer was allocated for.

See Also

MGL_copyToBuffer, MGL_copyBitmapToBuffer, MGL_updateBufferCache, MGL_updateFromBufferCache, MGL_putBuffer, MGL_putBufferSection, MGL_putBufferSrcTrans, MGL_putBufferSrcTransSection, MGL_putBufferDstTrans,

*MGL_putBufferDstTransSection, MGL_putBufferPatt, MGL_putBufferPattSection,
MGL_putBufferFx, MGL_putBufferFxSection, MGL_stretchBuffer, MGL_stretchBufferSection,
MGL_stretchBufferFx, MGL_stretchBufferFxSection*

MGL_putDivot

Replaces a divot of video memory to the location from which it was copied.

Declaration

```
void MGLAPI MGL_putDivot(  
    MGLDC *dc,  
    void *divot)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to restore the divot to
<i>divot</i>	Pointer to the divot to replace

Description

This function replaces a rectangle of video memory that was saved previously with the *MGL_getDivot* function. The divot is replaced at the same location that it was taken from on the current device context.

A divot is defined as being a rectangular area of video memory that you wish to save, however the bounding rectangle for the divot is expanded slightly to properly aligned boundaries for the absolute maximum performance with the current device context. This function is generally used to store the video memory behind pull down menus and pop up dialog boxes, and the memory can only be restored to exactly the same position that it was saved from.

See Also

MGL_divotSize

MGL_putIcon

Draw an icon at the specified location.

Declaration

```
void MGLAPI MGL_putIcon(
    MGLDC *dc,
    int x,
    int y,
    const icon_t *icon)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to draw icon on
<i>x</i>	x coordinate to draw icon at
<i>y</i>	y coordinate to draw icon at
<i>icon</i>	Icon to display

Description

Draws an icon at the specified location on the current device context. The icon may be in any color format, and will be translated as necessary to the color format of the display device context. The icon is drawn by punching a black hole in the background with the icon mask, and then OR'ing in the image bitmap for the icon.

See Also

MGL_loadBitmap, MGL_putBitmap, MGL_putBitmapSection, MGL_putBitmapSrcTrans, MGL_putBitmapSrcTransSection, MGL_putBitmapDstTrans, MGL_putBitmapDstTransSection, MGL_putBitmapMask, MGL_putBitmapPatt, MGL_putBitmapPattSection, MGL_putBitmapFx, MGL_putBitmapFxSection, MGL_stretchBitmap, MGL_stretchBitmapSection, MGL_stretchBitmapFx, MGL_stretchBitmapFxSection, MGL_putIcon

MGL_putMonoImage

Draw a monochrome bitmap at the specified location.

Declaration

```
void MGLAPI MGL_putMonoImage (
    MGLDC *dc,
    int x,
    int y,
    int width,
    int byteWidth,
    int height,
    void *image)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to draw bitmap on
<i>x</i>	x coordinate to draw bitmap at
<i>y</i>	y coordinate to draw bitmap at
<i>width</i>	Width of the bitmap to draw in pixels
<i>byteWidth</i>	Width of the bitmap image in bytes
<i>height</i>	Height of the bitmap in scanlines
<i>image</i>	Pointer to the buffer holding the bitmap

Description

This function draws a monochrome bitmap in the current foreground color on the current device context. Where a bit is a 1 in the bitmap definition, a pixel is plotted in the foreground color, where a bit is a 0 the original pixels are left alone. This function can be used to implement fast hardware pixel masking for drawing fast transparent bitmaps on devices that do not have a native hardware transparent BitBlt function.

MGL_quickInit

Quick initialisation function for first time MGL users.

Declaration

```
MGLDC * MGLAPI MGL_quickInit(
    int xRes,
    int yRes,
    int bitsPerPixel,
    int numPages)
```

Prototype In

mgraph.h

Parameters

<i>xRes</i>	Horizontal resolution for the display mode in pixels
<i>yRes</i>	Vertical resolution for the display mode in lines
<i>bitsPerPixel</i>	Color depth for the display mode
<i>numPages</i>	Number of display pages to use

Return Value

Pointer to a fullscreen display device context.

Description

This function is intended to help first time MGL users get up and running quickly. Using this utility function you can initialise the MGL for use in a fullscreen display mode with a single line of code. Note that this function will automatically bail out with an error message if any of the initialisation code fails, so you don't need to check for error conditions on return from this function.

A small 'Hello World' type application using the MGL might be coded as follows:

```
int main(void)
{
    MGLDC    *dc;
    event_t  evt;
    font_t   *font;

    dc = MGL_quickInit(640,480,8,1);
    if ((font = MGL_loadFont("pc8x8.fnt")) == NULL)
        MGL_fatalError(MGL_errorMsg(MGL_result()));
    MGL_useFont(font);
    MGL_drawStrXY(0,0,"Hello World!");
    EVT_halt(&evt,EVT_KEYDOWN);
    MGL_exit();
}
```

Note: *Once you are more familiar with the MGL, we highly recommend you add proper initialisation code to your program using MGL_init and related functions. Unless your application is running on a dedicated system, it is not usually a good idea to hard code a resolution and color depth into the application, but rather allow the user to select the mode to be used via configuration files or menus.*

Note: *This function*

See Also

MGL_init, MGL_exit

MGL_random

Generate a random 16-bit number between 0 and max.

Declaration

```
ushort MGLAPI MGL_random(  
    ushort max)
```

Prototype In

mgraph.h

Parameters

max Largest desired value

Return Value

Random 16-bit number between 0 and max.

See Also

MGL_randoml, *MGL_srand*

MGL_randoml

Generate a random 32-bit number between 0 and max.

Declaration

```
ulong MGLAPI MGL_randoml(  
    ulong max)
```

Prototype In

mgraph.h

Parameters

max Largest desired value

Return Value

Random 32-bit number between 0 and max.

See Also

MGL_random, *MGL_srand*

MGL_realColor

Returns the real packed MGL color for a color index.

Declaration

```
color_t MGLAPI MGL_realColor(  
    MGLDC *dc,  
    int color)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to map color with
<i>color</i>	Color to map

Return Value

Real packed MGL color value for the color index.

Description

This function returns a packed MGL color value appropriate for the specified device context given a color index. This routine works with all device contexts, including RGB device contexts. For the color index devices, the value is simply returned unchanged. However for RGB devices, the color index is translated via the current color palette for the device to find the appropriate packed MGL color value for that device. Thus you can still write code for RGB devices that works with color indexes (although you cannot do things like hardware palette fades and rotates as the palette is implemented in software).

See Also

MGL_setColorCI, MGL_setColorRGB, MGL_setPalette, MGL_getPalette

MGL_realizePalette

Realizes the hardware color palette for the display device context.

Declaration

```
void MGLAPI MGL_realizePalette(
    MGLDC *dc,
    int numColors,
    int startIndex,
    ibool waitVRT)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to realize palette for
<i>numColors</i>	Number of colors to realize
<i>startIndex</i>	Starting index of first color to realize
<i>waitVRT</i>	True if routine should sync to vertical retrace, false if not.

Description

This function realizes the hardware palette associated with a display device context. Calls to *MGL_setPalette* only update the palette values in the color palette for the device context structure, but do not actually program the hardware palette for display device contexts in 4 and 8 bits per pixel modes. In order to program the hardware palette you must call this routine.

When the hardware palette is realized, you normally need to sync to the vertical retrace to ensure that the palette values are programmed without the onset of snow (see *MGL_setPaletteSnowLevel* to adjust the number of colors programmed per retrace period). If however you wish to perform double buffered animation and change the hardware color palette at the same time, you should call this routine immediately after calling either *MGL_setVisualPage* or *MGL_swapBuffers* with the waitVRT flag set to false.

See Also

MGL_setPalette, *MGL_setVisualPage*, *MGL_swapBuffers*, *MGL_setPaletteSnowLevel*

MGL_rect

Draws a rectangle outline.

Declaration

```
void MGL_rect(  
    rect_t r)
```

Prototype In

mgraph.h

Parameters

r Rectangle to draw

Description

This function is the same as *MGL_rectCoord*, however it takes an entire rectangle as the parameter instead of coordinates.

See Also

MGL_rectCoord, *MGL_rectPt*

MGL_rectCoord

Draws a rectangle outline.

Declaration

```
void MGLAPI MGL_rectCoord(  
    int left,  
    int top,  
    int right,  
    int bottom)
```

Prototype In

mgraph.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

Description

This function draws a rectangle outline in the current drawing attributes at the specified location.

See Also

MGL_rect, *MGL_rectPt*

MGL_rectPt

Draws a rectangle outline.

Declaration

```
void MGL_rectPt(  
    point_t leftTop,  
    point_t rightBottom)
```

Prototype In

mgraph.h

Description

This function is the same as *MGL_rectCoord*, however it takes the top left and bottom right coordinates of the rectangle as two points instead of four coordinates.

See Also

MGL_rectCoord, *MGL_rect*

MGL_registerEventProc

Registers a user supplied window procedure for event handling.

Declaration

```
void MGLAPI MGL_registerEventProc(  
    MGL_WNDPROC userWndProc)
```

Prototype In

mglwin.h

Parameters

userWndProc Point to user supplied Window Procedure

Description

This function registers a user supplied window procedure with MGL that will be used for event handling purposes. By default MGL applications can simply use the EVT_* event handling functions that are common to both the DOS and Windows versions of MGL. However developers porting Windows specific code from DirectDraw may wish to use their existing window specific event handling code. This function allows you to do this by telling MGL to use your window procedure for all event processing.

See Also

EVT_getNext

MGL_registerFullScreenWindow

Registers a user window with the MGL to be used for fullscreen modes

Declaration

```
void MGLAPI MGL_registerFullScreenWindow(  
    MGL_HWND hwndFullScreen)
```

Prototype In

mglwin.c

Description

This function allows the application to create the window used for fullscreen modes, and let the MGL know about the window so it will use that window instead of creating it's own fullscreen window. By default when you create a fullscreen device context, the MGL will create a fullscreen window that covers the entire desktop that is used to capture Windows events such as keyboard events, mouse events and activation events. However in some situations it is beneficial to have only a single window that is used for all fullscreen graphics modes, as well as windows modes (primarily to be able to properly support DirectSound via a single window).

If you have registered a fullscreen window with the MGL, the MGL will take that window, zoom it fullscreen and modify the window styles and attributes to remove the title bar and other window decorations. When the MGL then returns from fullscreen mode back to GDI mode, it will restore the window back to the original position, style and state it was in before the fullscreen mode was created. This allows you to create a single window with the MGL and use it for both windowed modes and fullscreen modes.

Note: *When the MGL exits via a call to MGL exit, the fullscreen window that you passed in will be destroyed (necessary to work around bugs in DirectDraw).*

MGL_restoreAttributes

Restores a previously saved set of rasterizing attributes.

Declaration

```
void MGLAPI MGL_restoreAttributes(  
    attributes_t *attr)
```

Prototype In

mgraph.h

Parameters

attr Pointer to the attribute list to restore

Description

This function restores a set of attributes that were saved with the *MGL_getAttributes* routine. The attributes list represents the current state of the MGL. The value of the color palette is not changed by this routine.

Note: *Calling this function sets the viewport and clip rectangle for the device context to the values that were saved by*

See Also

MGL_getAttributes

MGL_result

Returns result code of the last graphics operation.

Declaration

```
int MGLAPI MGL_result(void)
```

Prototype In

mgraph.h

Return Value

Result code of the last graphics operation

Description

This function returns the result code of the last graphics operation. The internal result code is reset back to grOK on return from this routine, so you should only call the routine once after the graphics operation. Error codes returned by this function are enumerated in *MGL_errorType*.

See Also

MGL_setResult, *MGL_errorMsg*

MGL_resume

Resume low level event handling code.

Declaration

```
void MGLAPI MGL_resume(void)
```

Prototype In

mgldos.h

Description

Resumes the event handling code for MGL. This function should be used to re- enable the MGL event handling code after shelling out to DOS from your application code or running another application.

See Also

MGL_suspend, MGL_init

MGL_rgbColor

Computes a packed MGL color from a 24 bit RGB tuple.

Declaration

```
color_t MGLAPI MGL_rgbColor(
    MGLDC *dc,
    uint R,
    uint G,
    uint B)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to map color with
<i>R</i>	Red component of color to map (0 - 255)
<i>G</i>	Green component of color to map (0 - 255)
<i>B</i>	Blue component of color to map (0 - 255)

Return Value

Packed MGL color closest to specified RGB tuple.

Description

This function computes the packed MGL color value from a specific 24 bit RGB tuple for a device context. If the device context is an RGB device context or an 8 bit device in RGB dithered mode, this value simply returns the proper packed MGL pixel value representing this color (the same as *MGL_packColor* would). However if the device context is a color index device, the color palette is searched for the color value that is the closest to the specified color. This function allows you to specify a color given an RGB tuple, and will work in color index modes as well with any color palette.

See Also

MGL_realColor, *MGL_setColorCI*, *MGL_setColorRGB*

MGL_rgnEllipse

Generate an ellipse outline as a region.

Declaration

```
region_t * MGLAPI MGL_rgnEllipse(  
    rect_t extentRect,  
    const region_t *pen)
```

Prototype In

mgraph.h

Parameters

<i>extentRect</i>	Bounding rectangle for the ellipse
<i>pen</i>	Region to use as the _pen when drawing the ellipse

Return Value

New region generated, NULL if out of memory.

Description

This function generates the outline of an ellipse as a complex region by dragging the specified _pen region around the perimeter of the ellipse. The _pen used can be any arbitrary shape, however rectangular _pens are special cased to provide fast region generation.

See Also

MGL_rgnEllipseArc

MGL_rgnEllipseArc

Generate an elliptical arc outline as a region.

Declaration

```
region_t * MGLAPI MGL_rgnEllipseArc(
    rect_t extentRect,
    int startAngle,
    int endAngle,
    const region_t *pen)
```

Prototype In

mgraph.h

Parameters

<i>extentRect</i>	Bounding rectangle for the ellipse
<i>startAngle</i>	Starting angle for the elliptical arc
<i>endAngle</i>	Ending angle for the elliptical arc
<i>pen</i>	Region to use as the pen when drawing the ellipse

Return Value

New region generated, NULL if out of memory.

Description

This function generates the outline of an elliptical arc as a complex region by dragging the specified pen region around the perimeter of the ellipse. The pen used can be any arbitrary shape, however rectangular pens are special cased to provide fast region generation.

See Also

MGL_rgnEllipse, *MGL_rgnGetArcCoords*

MGL_rgnGetArcCoords

Returns the real arc coordinates for an elliptical arc region.

Declaration

```
void MGLAPI MGL_rgnGetArcCoords(  
    arc_coords_t *coords)
```

Prototype In

mgraph.h

Parameters

coords Pointer to structure to store coordinates

Description

This function returns the center coordinate, and starting and ending points on the ellipse that defines the last elliptical arc region that was generated. You can then use these coordinates to draw a line from the center of the ellipse to the starting and ending points to complete the outline of an elliptical wedge.

Note that you must call this routine immediately after calling the *MGL_rgnEllipseArc* routine.

See Also

MGL_rgnEllipseArc

MGL_rgnLine

Generate a line as a region.

Declaration

```
region_t *MGL_rgnLine(  
    point_t p1,  
    point_t p2,  
    const region_t *pen)
```

Prototype In

mgraph.h

Parameters

<i>p1</i>	First endpoint
<i>p2</i>	Second endpoint
<i>pen</i>	Region to use as the pen when drawing the line

Return Value

New region generated, NULL if out of memory.

Description

This function is the same as *MGL_rgnLine* but takes the parameters for the line as two points instead of coordinates.

See Also

MGL_rgnLineCoord

MGL_rgnLineCoord

Generate a line as a region.

Declaration

```
region_t * MGLAPI MGL_rgnLineCoord(  
    int x1,  
    int y1,  
    int x2,  
    int y2,  
    const region_t *pen)
```

Prototype In

mgraph.h

Parameters

<i>x1</i>	x coordinate for first endpoint
<i>y1</i>	y coordinate for first endpoint
<i>x2</i>	x coordinate for second endpoint
<i>y2</i>	y coordinate for second endpoint
<i>pen</i>	Region to use as the pen when drawing the line

Return Value

New region generated, NULL if out of memory.

Description

Generates a region as a line starting at the point (x1,y1) and ending at the point (x2,y2). Note that this function takes the coordinates of the lines in integer format.

See Also

MGL_rgnLine

MGL_rgnPolygon

Generates a solid complex polygonal region.

Declaration

```
region_t * MGLAPI MGL_rgnPolygon(
    int count,
    point_t *vArray,
    int vinc,
    int xOffset,
    int yOffset)
```

Prototype In

mgraph.h

Parameters

<i>count</i>	Number of vertices to draw
<i>vArray</i>	Array of vertices
<i>vinc</i>	Increment to get to next vertex
<i>xOffset</i>	Offset of X coordinates
<i>yOffset</i>	Offset of Y coordinates

Description

This function generates a complex region that represents a convex polygon. A “convex” polygon is defined as a polygon such that every horizontal line drawn through the polygon would cross exactly two active edges (neither horizontal lines nor zero-length edges count as active edges; both are acceptable anywhere in the polygon). Right and left edges may cross (polygons may be nonsimple). Attempting to scan convert a polygon that does not fit this description will produce unpredictable results.

Note: All vertices are offset by (xOffset,yOffset) and are in regular integer format.

See Also

MGL_rgnPolygonCnvxFX, *MGL_rgnPolygonFX*

MGL_rgnPolygonCnvx

Generates a solid convex polygonal region.

Declaration

```
region_t * MGLAPI MGL_rgnPolygonCnvx (
    int count,
    point_t *vArray,
    int vinc,
    int xOffset,
    int yOffset)
```

Prototype In

mgraph.h

Parameters

<i>count</i>	Number of vertices to draw
<i>vArray</i>	Array of vertices
<i>vinc</i>	Increment to get to next vertex
<i>xOffset</i>	Offset of X coordinates
<i>yOffset</i>	Offset of Y coordinates

Description

This function generates a complex region that represents a convex polygon. A “convex” polygon is defined as a polygon such that every horizontal line drawn through the polygon would cross exactly two active edges (neither horizontal lines nor zero-length edges count as active edges; both are acceptable anywhere in the polygon). Right and left edges may cross (polygons may be nonsimple). Attempting to scan convert a polygon that does not fit this description will produce unpredictable results.

Note: *All vertices are offset by (xOffset,yOffset) and are in regular integer format.*

See Also

MGL_rgnPolygonCnvx, MGL_rgnPolygon

MGL_rgnPolygonCnvxFX

Generates a solid convex polygonal region.

Declaration

```
region_t * MGLAPI MGL_rgnPolygonCnvxFX(
    int count,
    fxpoint_t *vArray,
    int vinc,
    fix32_t xOffset,
    fix32_t yOffset)
```

Prototype In

mgraph.h

Parameters

<i>count</i>	Number of vertices to draw
<i>vArray</i>	Array of vertices
<i>vinc</i>	Increment to get to next vertex
<i>xOffset</i>	Offset of X coordinates
<i>yOffset</i>	Offset of Y coordinates

Description

This function generates a complex region that represents a convex polygon. A “convex” polygon is defined as a polygon such that every horizontal line drawn through the polygon would cross exactly two active edges (neither horizontal lines nor zero-length edges count as active edges; both are acceptable anywhere in the polygon). Right and left edges may cross (polygons may be nonsimple). Attempting to scan convert a polygon that does not fit this description will produce unpredictable results.

Note: All vertices are offset by (xOffset,yOffset) and are in 16.16 fixed point format.

See Also

MGL_rgnPolygonCnvx, *MGL_rgnPolygon*

MGL_rgnPolygonFX

Generates a solid complex polygonal region.

Declaration

```
region_t * MGLAPI MGL_rgnPolygonFX(
    int count,
    fxpoint_t *vArray,
    int vinc,
    fix32_t xOffset,
    fix32_t yOffset)
```

Prototype In

mgraph.h

Parameters

<i>count</i>	Number of vertices to draw
<i>vArray</i>	Array of vertices
<i>vinc</i>	Increment to get to next vertex
<i>xOffset</i>	Offset of X coordinates
<i>yOffset</i>	Offset of Y coordinates

Description

This function generates a complex region that represents a convex polygon. A “convex” polygon is defined as a polygon such that every horizontal line drawn through the polygon would cross exactly two active edges (neither horizontal lines nor zero-length edges count as active edges; both are acceptable anywhere in the polygon). Right and left edges may cross (polygons may be nonsimple). Attempting to scan convert a polygon that does not fit this description will produce unpredictable results.

Note: All vertices are offset by (xOffset,yOffset) and are in 16.16 fixed point format.

See Also

MGL_rgnPolygonCnvxFX, *MGL_rgnPolygon*

MGL_rgnSolidEllipse

Generates a solid ellipse as a region.

Declaration

```
region_t * MGLAPI MGL_rgnSolidEllipse(  
    rect_t extentRect)
```

Prototype In

mgraph.h

Parameters

extentRect Bounding rectangle for the ellipse

Return Value

New region generated, NULL if out of memory.

Description

This function generates a solid ellipse as a complex region

See Also

MGL_rgnSolidEllipseArc

MGL_rgnSolidEllipseArc

Generates a solid elliptical arc as a region.

Declaration

```
region_t * MGLAPI MGL_rgnSolidEllipseArc(  
    rect_t extentRect,  
    int startAngle,  
    int endAngle)
```

Prototype In

mgraph.h

Parameters

<i>extentRect</i>	Bounding rectangle for the ellipse
<i>startAngle</i>	Starting angle for the elliptical arc
<i>endAngle</i>	Ending angle for the elliptical arc

Return Value

New region generated, NULL if out of memory.

Description

This function generates a solid elliptical arc as a region.

See Also

MGL_rgnEllipse, *MGL_rgnGetArcCoords*

MGL_rgnSolidRect

Generate a solid rectangle as a region from two points.

Declaration

```
void MGL_rgnSolidRect(  
    rect_t r)
```

Prototype In

mgraph.h

Parameters

r Rectangle the coordinates of the region

Return Value

New region generated, NULL if out of memory.

Description

This function generates a solid rectangle as a region.

See Also

MGL_rgnSolidRectCoord, *MGL_rgnSolidRectPt*

MGL_rgnSolidRectCoord

Generate a solid rectangle as a region.

Declaration

```
region_t * MGLAPI MGL_rgnSolidRectCoord(
    int left,
    int top,
    int right,
    int bottom)
```

Prototype In

mgraph.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

This function generates a solid rectangle as a region.

See Also

MGL_rgnSolidRect, *MGL_rgnSolidRectPt*

MGL_rgnSolidRectPt

Generate a solid rectangle as a region from two points.

Declaration

```
region_t MGL_rgnSolidRectPt(  
    point_t lt,  
    point_t rb)
```

Prototype In

mgraph.h

Parameters

<i>lt</i>	Point containing left-top coordinates of the region
<i>rb</i>	Point containing right-bottom coordinates of the region

Return Value

New region generated, NULL if out of memory.

Description

This function generates a solid rectangle as a region.

See Also

MGL_rgnSolidRectCoord, *MGL_rgnSolidRect*

MGL_rightBottom

Returns the bottom right coordinate from a rectangle.

Declaration

```
point_t MGL_rightBottom(  
    rect_t r)
```

Prototype In

mgraph.h

Parameters

r Rectangle to get coordinate from

Return Value

The bottom right coordinate of the rectangle.

Description

Returns the bottom right coordinates from a rectangle.

See Also

MGL_leftTop

MGL_rotateGlyph

Declaration

```
void MGLAPI MGL_rotateGlyph(
    uchar *dst,
    uchar *src,
    int *byteWidth,
    int *height,
    int rotation)
```

Prototype In

mgraph.h

Parameters

<i>dst</i>	Destination glyph buffer
<i>src</i>	Source glyph buffer
<i>byteWidth</i>	Width of the glyph in bytes
<i>height</i>	Height of the glyph in scanlines
<i>rotation</i>	Rotation direction for the glyph

Description

This function computes the rotated image glyph of the source glyph, and stores the value in the destination buffer. The source buffer is not modified, and the rotated image glyph may possibly be larger than the source glyph. The resulting width and height of the destination glyph is returned. Supported directions are enumerated in *MGL_textDirType*

Note: *You must preallocate enough space to hold the rotated glyph in the destination buffer, as this may actually be larger than the source glyph.*

The final size will be the following:

```
(height + 7) / 8 + byteWidth * 8
```

MGL_rotatePalette

Rotates the palette values for a device context.

Declaration

```
void MGLAPI MGL_rotatePalette(
    MGLDC *dc,
    int numColors,
    int startIndex,
    int direction)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context containing palette
<i>numColors</i>	Number of colors to rotate
<i>startIndex</i>	Starting index for colors to rotate
<i>direction</i>	Direction to rotate the palette entries

Description

This function rotates the palette values in the device context in the specified direction. Note that this routine does not effect the currently active hardware palette, and you must call *MGL_realizePalette* in order to make the program the rotated palette to the hardware.

Supported directions of rotation are enumerated in *MGL_palRotateType*.

When the direction specified is *MGL_ROTATE_UP*, the first entry in the palette is moved to the last position in the palette, and all the remaining entries are moved one position up in the array. If the direction specified is *MGL_ROTATE_DOWN*, the last entry is moved into the first entry of the palette, and the remaining entries are all moved one position down in the array.

See Also

MGL_setPalette, *MGL_getPalette*, *MGL_fadePalette*

MGL_saveBitmapFromDC

Save a portion of a device context to bitmap file on disk.

Declaration

```
ibool MGLAPI MGL_saveBitmapFromDC (
    MGLDC *dc,
    const char *bitmapName,
    int left,
    int top,
    int right,
    int bottom)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to save
<i>bitmapName</i>	Name of bitmap file to save
<i>left</i>	Left coordinate of bitmap to save
<i>top</i>	Top coordinate of bitmap to save
<i>right</i>	Right coordinate of bitmap to save
<i>bottom</i>	Bottom coordinate of bitmap to save

Return Value

True on success, false on error.

Description

This function saves a portion of a device context as a bitmap file to disk. If this function fails for some reason, it will return false and you can get the error code from the *MGL_result* function.

Note that the source rectangle for the bitmap to be saved is not clipped to the current clipping rectangle for the device context, but it is mapped to the current viewport. If you specify dimensions that are larger than the current device context, you will get garbage in the bitmap file as a result.

See Also

MGL_loadBitmap, *MGL_loadBitmapIntoDC*

MGL_saveJPEGFromDC

Save a portion of a device context to JPEG on disk.

Declaration

```
ibool MGLAPI MGL_saveJPEGFromDC (
    MGLDC *dc,
    const char *JPEGName,
    int left,
    int top,
    int right,
    int bottom,
    int quality)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to save
<i>JPEGName</i>	Name of bitmap file to save
<i>left</i>	Left coordinate of bitmap to save
<i>top</i>	Top coordinate of bitmap to save
<i>right</i>	Right coordinate of bitmap to save
<i>bottom</i>	Bottom coordinate of bitmap to save
<i>quality</i>	Quality factor for compression (1-100)

Return Value

True on success, false on error.

Description

This function saves a portion of a device context as a JPEG format bitmap file to disk. If this function fails for some reason, it will return false and you can get the error code from the *MGL_result* function.

Note that the source rectangle for the bitmap to be saved is not clipped to the current clipping rectangle for the device context, but it is mapped to the current viewport. If you specify dimensions that are larger than the current device context, you will get garbage in the bitmap file as a result.

Note that MGL currently only supports saving bitmap data to JPEG files from 8 bits per pixel device contexts.

See Also

MGL_LoadJPEG, *MGL_loadJPEGIntoDC*

MGL_savePCXFromDC

Save a portion of a device context to PCX on disk.

Declaration

```
ibool MGLAPI MGL_savePCXFromDC (
    MGLDC *dc,
    const char *PCXName,
    int left,
    int top,
    int right,
    int bottom)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to save
<i>PCXName</i>	Name of bitmap file to save
<i>left</i>	Left coordinate of bitmap to save
<i>top</i>	Top coordinate of bitmap to save
<i>right</i>	Right coordinate of bitmap to save
<i>bottom</i>	Bottom coordinate of bitmap to save

Return Value

True on success, false on error.

Description

This function saves a portion of a device context as a PCX format bitmap file to disk. If this function fails for some reason, it will return false and you can get the error code from the *MGL_result* function.

Note that the source rectangle for the bitmap to be saved is not clipped to the current clipping rectangle for the device context, but it is mapped to the current viewport. If you specify dimensions that are larger than the current device context, you will get garbage in the bitmap file as a result.

Note that MGL currently only supports saving bitmap data to PCX files from 8 bits per pixel device contexts.

See Also

MGL_LoadPCX, *MGL_loadPCXIntoDC*

MGL_savePNGFromDC

Save a portion of a device context to a PNG file on disk.

Declaration

```
ibool MGLAPI MGL_savePNGFromDC (
    MGLDC *dc,
    const char *PNGName,
    int left,
    int top,
    int right,
    int bottom)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to save
<i>PNGName</i>	Name of bitmap file to save
<i>left</i>	Left coordinate of bitmap to save
<i>top</i>	Top coordinate of bitmap to save
<i>right</i>	Right coordinate of bitmap to save
<i>bottom</i>	Bottom coordinate of bitmap to save

Return Value

True on success, false on error.

Description

This function saves a portion of a device context as a PNG format bitmap file to disk. If this function fails for some reason, it will return false and you can get the error code from the *MGL_result* function.

This function supports saving in native formats of 1, 4, 8, 24, and 32 bits. Images with bit depths of 15 or 16 bits will be expanded to 24 bits.

Note that the source rectangle for the bitmap to be saved is not clipped to the current clipping rectangle for the device context, but it is mapped to the current viewport. If you specify dimensions that are larger than the current device context, you will get garbage in the bitmap file as a result.

See Also

MGL_savePNGFromDCExt, *MGL_loadPNG*, *MGL_loadPNGIntoDC*

MGL_savePNGFromDCExt

Save a portion of a device context to a PNG file on disk.

Declaration

```
ibool MGLAPI MGL_savePNGFromDCExt (
    MGLDC *dc,
    const char *PNGName,
    int left,
    int top,
    int right,
    int bottom,
    ibool savePalette)
```

Prototype In
mgraph.h

Parameters

<i>dc</i>	Device context to save
<i>PNGName</i>	Name of bitmap file to save
<i>left</i>	Left coordinate of bitmap to save
<i>top</i>	Top coordinate of bitmap to save
<i>right</i>	Right coordinate of bitmap to save
<i>bottom</i>	Bottom coordinate of bitmap to save
<i>savePalette</i>	True to save palette, false to skip it

Return Value

True on success, false on error.

Description

This function saves a portion of a device context as a PNG format bitmap file to disk. If this function fails for some reason, it will return false and you can get the error code from the *MGL_result* function.

This function supports saving in native formats of 1, 4, 8, 24, and 32 bits. Images with bit depths of 15 or 16 bits will be expanded to 24 bits. It also supports saving 8-bit + alpha bitmaps as 8-bit grayscale + alpha PNG files. The PNG specification itself does not support saving the palette information, however in practice you can save a palette to this bitmap format and have the bitmap load and display correctly in most PNG viewers. MGL can also load those bitmaps properly from disk as well. If you pass a value of false to 'savePalette' the palette will not be saved in the PNG image (ie: resulting image is grayscale).

Note that the source rectangle for the bitmap to be saved is not clipped to the current clipping rectangle for the device context, but it is mapped to the current viewport. If you specify dimensions that are larger than the current device context, you will get garbage in the bitmap file as a result.

See Also

MGL_loadPNG, MGL_loadPNGIntoDC

MGL_scanLine

Fills a specified scanline with no clipping, in screen space.

Declaration

```
void MGLAPI MGL_scanLine(
    int y,
    int x1,
    int x2)
```

Prototype In

mgraph.h

Parameters

<i>y</i>	y coordinate of scanline to fill
<i>x1</i>	Starting x coordinate of scanline to fill
<i>x2</i>	Ending x coordinate of scanline to fill

Description

MGL_scanLine fills the specified portion of a scanline in the current attributes and fill pattern. This can be used to implement higher level complex fills, such as region fills, floodfills etc.

Note: *This function is intended as a low level building block, and as such does not do any clipping and takes coordinates in screen space directly. If you want to draw a clipped, viewport relative scanline use the MGL_lineCoord family of functions.*

See Also

MGL_penStyleType, MGL_setPenBitmapPattern, MGL_setPenPixmapPattern, MGL_lineCoord

MGL_sectRect

Compute the intersection between two rectangles.

Declaration

```
ibool MGLAPI MGL_sectRect (  
    rect_t r1,  
    rect_t r2,  
    rect_t *d)
```

Prototype In

mgraph.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

MGL_sectRectFast, *MGL_sectRectCoord*, *MGL_unionRect*

MGL_sectRectCoord

Compute the intersection between two rectangles.

Declaration

```
ibool MGLAPI MGL_sectRectCoord(
    int left1,
    int top1,
    int right1,
    int bottom1,
    int left2,
    int top2,
    int right2,
    int bottom2,
    rect_t *d)
```

Prototype In

mgraph.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

MGL_sectRectFastCoord, *MGL_sectRect*, *MGL_unionRect*

MGL_sectRectFast

Compute the intersection between two rectangles.

Declaration

```
void MGL_sectRectFast(  
    rect_t r1,  
    rect_t r2,  
    rect_t *d)
```

Prototype In

mgraph.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 *MGL_sectRect* but it is implemented as a macro and does not test the rectangle for intersection.

See Also

MGL_sectRect, *MGL_sectRectCoord*, *MGL_unionRect*

MGL_sectRectFastCoord

Compute the intersection between two rectangles.

Declaration

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

Prototype In

mgraph.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

MGL_sectRectFastCoord, *MGL_sectRect*, *MGL_unionRect*

MGL_sectRegion

Compute the Boolean intersection between two regions.

Declaration

```
region_t * MGLAPI MGL_sectRegion(  
    const region_t *r1,  
    const region_t *r2)
```

Prototype In

mgraph.h

Parameters

r1 First region to compute intersection with
r2 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

MGL_diffRegion, *MGL_unionRegion*, *MGL_sectRegionRect*

MGL_sectRegionRect

Compute the Boolean intersection between a region and a rectangle.

Declaration

```
region_t * MGLAPI MGL_sectRegionRect (
    const region_t *r1,
    const rect_t *r2)
```

Prototype In

mgraph.h

Parameters

r1 Region to compute intersection with
r2 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 *MGL_sectRegion* with a simple region as the second region to intersect.

See Also

MGL_sectRegion, *MGL_diffRegion*, *MGL_unionRegion*

MGL_selectDisplayDevice

Selects the specified display device as the active device.

Declaration

```
int MGLAPI MGL_selectDisplayDevice(
    int device)
```

Prototype In

mgraph.h

Parameters

device New display device index to make active

Return Value

Index of previously active device, or -1 on failure.

Description

This function is used to select one of the attached display devices as the active display device from this point forward. This function is used to change the active display device after a call to *MGL_init*, and all subsequent calls to device initialisation functions work with the new active device. 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. The maximum number of display devices is defined by the compile time constant *MAX_DISPLAY_DEVICES*.

You can also select a 'mixed' mode of operation by passing in the *MM_MODE_MIXED* parameter, which fully enables the primary display controller but only enables the secondary display controllers relocateable memory mapped regions. If this mode is selected, you must have first initialised all attached secondary controllers to a graphics display mode or this call will fail.

Note: *Some older display controllers cannot support mixed mode as the VGA compatible resources cannot be disabled. These controllers will still work fine, however there is more overhead involved in the calls to *MGL_makeCurrentDC* as this function must switch the active display device every time it is called. If mixed mode is enabled, the *MGL_makeCurrentDC* function does not have any extra overhead for multiple display controller support.*

The general function call sequence to use multiple controllers in the MGL is as follows:

```
int      numDevices, device, mode;
MGLDC    *dc[MAX_DISPLAY_DEVICES];

// Initialise the library
if ((numDevices = MGL_init(".", NULL)) == 0)
    MGL_fatalError(MGL_errorMsg(MGL_result()));
for (device = 0; device < numDevices; device++) {
```

```

    MGL_selectDisplayDevice(device);
    if ((mode = MGL_findMode(640,480,8)) == -1)
        MGL_fatalError(MGL_errorMsg(MGL_result()));
    dc[device] = MGL_createDisplayDC(mode,1,MGL_DEFAULT_REFRESH);
    if (!dc[device])
        MGL_fatalError(MGL_errorMsg(MGL_result()));
}
MGL_selectDisplayDevice(MM_MODE_MIXED);

// Now draw something on each device
for (device = 0; device < numDevices; device++) {
    MGL_makeCurrentDC(dc[0]);
    MGL_line(0,0,10,10);
}

// Now close down the MGL
MGL_exit();

```

Note: *Once you have initialised the display and created a display device context for each of the displays in the system, you do not need to call this function to switch the active display from within your rendering code. This is handled automatically by the MGL_makeCurrentDC function every time you switch the active MGL device context.*

See Also

MGL_init, MGL_enableAllDrivers, MGL_disableDriver, MGL_availablePages, MGL_modeResolution, MGL_findMode, MGL_addCustomMode, MGL_createDisplayDC, MGL_exit, MGL_result

MGL_setActivePage

Sets the currently active hardware display page for a display device context.

Declaration

```
void MGLAPI MGL_setActivePage(
    MGLDC *dc,
    int page)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Display device context of interest
<i>page</i>	Number of active hardware display page to use

Description

This function sets the currently active hardware video page to which all output from MGL is sent to. The first hardware video page is number 0, the second is 1 and so on. The number of available hardware video pages depends on the type of underlying hardware, the video mode resolution and amount of video memory installed. Thus not all video modes support multiple hardware video pages.

See Also

MGL_getActivePage, MGL_setVisualPage, MGL_getVisualPage, MGL_swapBuffers

MGL_setAlphaValue

Sets the current constant alpha value for blending functions.

Declaration

```
void MGLAPI MGL_setAlphaValue(  
    uchar alpha)
```

Prototype In

mgraph.h

Parameters

<i>alpha</i>	New constant alpha value for blending function, between 0 and 255
--------------	---

Description

This function sets the current constant alpha value used for pixel blending functions in the MGL. The constant alpha value is only used if one of the source or destination blending functions is set to include constant alpha. The constant alpha value should be in the range of 0 through 255.

See Also

MGL_setSrcBlendFunc, MGL_setDstBlendFunc, *MGL_getAlphaValue*

MGL_setAspectRatio

Sets the current video mode's aspect ratio.

Declaration

```
void MGLAPI MGL_setAspectRatio(
    int aspectRatio)
```

Prototype In

mgraph.h

Parameters

aspectRatio New value for the aspect ratio

Description

This function sets the aspect ratio for the device context to a new value. This ratio is equal to:

$$\frac{\text{pixel x size}}{\text{pixel y size}} * 1000$$

The device context aspect ratio can be used to display circles and squares on the device by approximating them with ellipses and rectangles of the appropriate dimensions. Thus in order to determine the number of pixels in the y direction for a square with 100 pixels in the x direction, we can simply use the code:

```
y_pixels = ((long)x_pixels * 1000) / aspectratio
```

Note the cast to a long to avoid arithmetic overflow, as the aspect ratio is returned as an integer value with 1000 being a 1:1 aspect ratio.

See Also

MGL_getAspectRatio

MGL_setBackColor

Sets the currently active background color.

Declaration

```
void MGLAPI MGL_setBackColor(  
    color_t color)
```

Prototype In

mgraph.h

Parameters

color New background color value

Description

Sets the current background color value. The background color value is used to clear the display and viewport with the *MGL_clearDevice* and *MGL_clearViewport* routines, and is also used for filling solid primitives in the MGL_BITMAP_OPAQUE fill mode.

Note that the value passed to this routine is either a color index or a color value in the correct packed pixel format for the device context. Use the *MGL_packColor* routine to pack 24 bit RGB values for RGB device contexts.

See Also

MGL_getBackColor, *MGL_setColor*, *MGL_getColor*, *MGL_packColor*

MGL_setBackMode

Sets the current background mode for monochrome bitmap and stipple line rendering.

Declaration

```
void MGLAPI MGL_setBackMode(  
    int mode)
```

Prototype In

mgraph.h

Parameters

mode New background mode to set

Description

This function sets the current background mode for the device context. Supported modes are enumerated in *MGL_backModes*. By default the MGL starts up in transparent mode, however you can change the background mode to draw monochrome bitmaps and stippled lines with all 0 bits draw in the background color instead of being transparent. This affects all monochrome bitmap functions (including text rendering) and stipple line functions.

See Also

MGL_getBackMode

MGL_setBlendFunc

Sets the current source and destination pixel blending function.

Declaration

```
void MGLAPI MGL_setBlendFunc (
    int srcBlendFunc,
    int dstBlendFunc)
```

Prototype In

mgraph.h

Parameters

<i>srcBlendFunc</i>	New source blending function, defined by <i>MGL_blendFuncType</i> .
<i>dstBlendFunc</i>	New destination blending function, defined by <i>MGL_blendFuncType</i> .

Description

This function sets the current source and destination pixel blending function used to enable pixel blending in the MGL. By default pixel blending is disabled, and the blending function is set to *MGL_BLEND_NONE*. Essentially the source and destination blend function define how to combine the source and destination pixel colors together to get the resulting destination color during rendering. For a more detailed description of the blending functions, see the documentation for the *MGL_blendFuncType* enumeration.

Note: *Blending is only enabled when both the source and destination blending functions are set to values other than MGL_BLEND_NONE.*

Note: *Blending and logical write modes are not supported at the same time. When you enable blending modes, the logical write mode set by MGL_setWriteMode is ignored.*

Note: *No hardware currently supports arbitrary pixel blending operations for 2D operations, so enabling blending modes causes the MGL to run entirely in software. Keep this in mind since if performance is important!*

See Also

MGL_getBlendFunc, MGL_setAlphaValue

MGL_setBlueCodeIndex

Sets the color index used for the blue code stereo sync mechanism

Declaration

```
void MGLAPI MGL_setBlueCodeIndex(  
    int index)
```

Prototype In

mgraph.h

Description

This function sets the color index used in the blue code stereo sync mechanism. The blue code system as pioneered by StereoGraphics, requires the MGL to draw pure blue sync lines at the bottom of the screen of differing lengths to signal to the LC glasses which is the left and right eye images. In order to do this in 8bpp color index modes, the MGL must take up a single palette entry for drawing the blue codes, which by default is set to index 255. You can use this function to set the blue code index to another value other than 255 to suit your applications palette arrangement.

Note: *You must call this function before you create a stereo display device context if you wish to change the blue code index.*

See Also

MGL_startStereo, MGL_stopStereo, MGL_createStereoDisplayDC, MGL_setStereoSyncType

MGL_setBufSize

Sets the size of the internal MGL buffer.

Declaration

```
void MGLAPI MGL_setBufSize(  
    unsigned size)
```

Prototype In

mgraph.h

Parameters

size New size of the internal MGL buffer

Description

This function sets the size of the internal MGL scratch buffer, which the MGL uses for local scratch space in various places. The default size of this buffer is 32Kb, which is adequate for most needs. If however you attempt to render some primitives and MGL runs out of local storage space you will need to increase the size of this internal buffer.

If you are running on an embedded system and need to trim the amount of memory used by the MGL, you may want to use this function to set a smaller buffer that is suitable for your application needs to decrease the memory footprint used by the MGL at runtime.

Note that this routine must be called before *MGL_init* is called for the first time.

See Also

MGL_init, *MGL_setMaxScanLineWidth*

MGL_setClipRect

Sets the current clipping rectangle.

Declaration

```
void MGLAPI MGL_setClipRect(  
    rect_t clip)
```

Prototype In

mgraph.h

Parameters

clip New clipping rectangle to be used

Description

Sets the current clipping rectangle coordinates. The current clipping rectangle is used to clip all output, and is always defined as being relative to the currently active viewport. The clipping rectangle can be no larger than the currently active viewport, and will be truncated if an attempt is made to allow clipping outside of the active viewport.

Note: *Setting a new clip rectangle with this function clears any existing complex clip region in the device context.*

See Also

MGL_setClipRectDC, MGL_getClipRect, MGL_setViewport, MGL_getViewport, MGL_setClipRegion

MGL_setClipRectDC

Sets the current clipping rectangle for a specific DC.

Declaration

```
void MGLAPI MGL_setClipRectDC(
    MGLDC *dc,
    rect_t clip)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Display device context in which the rectangle is located .
<i>clip</i>	New clipping rectangle to be used

Description

This function is the same as *MGL_setClipRect*, however the device context does not have to be the current device context.

Note: *Setting a new clip rectangle with this function clears any existing complex clip region in the device context.*

See Also

MGL_setClipRect, *MGL_getClipRect*, *MGL_setViewport*, *MGL_getViewport*, *MGL_setClipRegion*

MGL_setClipRegion

Sets the current complex clipping region.

Declaration

```
void MGLAPI MGL_setClipRegion(  
    region_t *region)
```

Prototype In

mgraph.h

Description

This function sets the current complex clipping region for the current device context. Complex clip regions are defined as unions of rectangles, and allow all rendering functions in the MGL to be clipped to arbitrary regions on the screen. Setting a complex clip region override the current setting for a simple clip rectangle. The complex clip region is used to clip all output, and is always defined as being relative to the currently active viewport. The clipping region can be no larger than the currently active viewport, and will be truncated if an attempt is made to allow clipping outside of the active viewport.

See Also

MGL_setClipRegionDC, MGL_getClipRegion, MGL_setViewport, MGL_getViewport, MGL_setClipRect

MGL_setClipRegionDC

Sets the current complex clipping region for a specific DC.

Declaration

```
void MGLAPI MGL_setClipRegionDC(
    MGLDC *dc,
    region_t *region)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Display device context in which the region is located .
<i>region</i>	New complex clipping region to be used

Description

This function sets the current complex clipping region for the specified device context. Complex clip regions are defined as unions of rectangles, and allow all rendering functions in the MGL to be clipped to arbitrary regions on the screen. Setting a complex clip region override the current setting for a simple clip rectangle. The complex clip region is used to clip all output, and is always defined as being relative to the currently active viewport. The clipping region can be no larger than the currently active viewport, and will be truncated if an attempt is made to allow clipping outside of the active viewport.

See Also

MGL_setClipRegion, *MGL_getClipRegion*, *MGL_setViewport*, *MGL_getViewport*, *MGL_setClipRect*

MGL_setColor

Sets the current foreground color.

Declaration

```
void MGLAPI MGL_setColor(  
    color_t color)
```

Prototype In

mgraph.h

Parameters

color New foreground color value

Description

Sets the current foreground color values. The foreground color value is used to draw all primitives.

Note that the value passed to this routine is either a color index or a color value in the correct packed pixel format for the current video mode. Use the *MGL_packColor* routine to pack 24 bit RGB values for direct color video modes.

See Also

MGL_getColor, *MGL_setBackColor*, *MGL_getBackColor*, *MGL_packColor*

MGL_setColorCI

Sets the current foreground color given a color index.

Declaration

```
void MGLAPI MGL_setColorCI(  
    int index)
```

Prototype In

mgraph.h

Parameters

index Color index of color to set

Description

Sets the current foreground color value given a color index. This routine works with all device contexts, including RGB device contexts. For the color index devices, the value for the foreground color is simply set unchanged. However for RGB devices, the color index is translated via the current color palette for the device to find the appropriate packed MGL color value for that device. Thus you can still write code for RGB devices that works with color indexes.

See Also

MGL_setColor, *MGL_setColorRGB*, *MGL_realColor*

MGL_setColorRGB

Sets the current foreground color given a 24 bit RGB tuple.

Declaration

```
void MGLAPI MGL_setColorRGB(  
    uchar R,  
    uchar G,  
    uchar B)
```

Prototype In

mgraph.h

Parameters

<i>R</i>	Red component of color (0 - 255)
<i>G</i>	Green component of color (0 - 255)
<i>B</i>	Blue component of color (0 - 255)

Description

This function sets the foreground color to a specific 24 bit RGB tuple. If the device context is an RGB device context or an 8 bit device in RGB dithered mode, this value simply sets the proper packed MGL pixel value representing this color (the same as *MGL_packColor* would). However if the device context is a color index device, the color palette is searched for the color value that is the closest to the specified color. This function allows you to specify a color given an RGB tuple, and will work in color index modes as well with any color palette.

See Also

MGL_setColor, *MGL_setColorCI*, *MGL_rgbColor*

MGL_setDefaultPalette

Resets the palette to the default values.

Declaration

```
void MGLAPI MGL_setDefaultPalette(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context of interest

Description

Sets the palette to the current MGL default values for the device context. This can be used to reset the palette to the original default values that the palette is programmed with when MGL is initialized.

See Also

MGL_getDefaultPalette, MGL_setPalette, MGL_getPalette

MGL_setDisplayStart

Changes the display start address for virtual scrolling.

Declaration

```
void MGLAPI MGL_setDisplayStart(
    MGLDC *dc,
    int x,
    int y,
    int waitVRT)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Scrolling display device context to change
<i>x</i>	New display start x coordinate
<i>y</i>	New display start y coordinate
<i>waitVRT</i>	Wait for retrace flag (<i>MGL_waitVRTFlagType</i>)

Description

This function sets the CRTC display starting address for the hardware scrolling device context to the specified (x,y) coordinate. You can use this routine to implement hardware scrolling or panning by moving the display start address coordinates.

The waitVRT flag is used for synchronizing with the vertical retrace and can be one of the following values:

<i>waitVRT</i>	Meaning
<i>MGL_waitVRT</i>	Set coordinates and update hardware, waiting for a vertical retrace during the update for flicker free panning.
<i>MGL_dontWait</i>	Set coordinates and update hardware, but do not wait for a vertical retrace when changing the hardware start address.
<i>minus1</i>	Set coordinates but don't update hardware display start.

Passing a waitVRT flag of -1 can be used to implement double buffering and hardware scrolling at the same time. To do this you would call this function first to set the display start x and y coordinates without updating the hardware, and then call *MGL_setVisualPage* to swap display pages and the new hardware start address will then be programmed.

Note: This function does not allow the *MGL_tripleBuffer* flag to be passed in. If you are doing triple buffering with virtual scrolling, you must pass a value of -1 in the waitVRT parameter, and then pass a value of *MGL_tripleBuffer* in the waitVRT parameter of the *MGL_setVisualPage* function.

See Also

MGL_getDisplayStart, *MGL_createScrollingDC*

MGL_setDitherMode

Sets the current dithering mode for all blitting operations.

Declaration

```
void MGLAPI MGL_setDitherMode(  
    int mode)
```

Prototype In

mgraph.h

Parameters

mode New dithering mode to make active

Description

This function sets the current dithering mode used when blitting RGB images to 8, 15 and 16 bits per pixel device context. If dithering is enabled, the blit operation will dither the resulting image to produce the best quality. When dithering is disabled, the blit operation uses the closest color which has less quality but is faster.

Note: *The closest color method is fastest when the destination device context is a 15 or 16 bits per pixel bitmap. However when the destination is an 8 bits per pixel device context, the dithering mode will usually be faster.*

Note: *Dithering is on by default in the MGL.*

See Also

MGL_getDitherMode

MGL_setDotsPerInch

Sets the outline font dots per inch value.

Declaration

```
void MGLAPI MGL_setDotsPerInch(  
    int xDPI,  
    int yDPI)
```

Prototype In

mgraph.h

Parameters

<i>xDPI</i>	Horizontal dots per inch
<i>yDPI</i>	Vertical dots per inch

Description

This function set the horizontal and vertical dots per inch factor, which is used to generate outline font bitmap glyphs. Making the value larger results in larger font bitmaps. The default value for PC compatible systems is 96 DPI.

MGL_setFileIO

Overrides the default file I/O functions used by MGL.

Declaration

```
void MGLAPI MGL_setFileIO(  
    fileio_t *fio)
```

Prototype In

mgraph.h

Parameters

fio Structure containing new file I/O functions

Description

This function allows the programmer to override the default file I/O functions used by all the MGL functions that access files (bitmap, font, icon and cursor loading). By default the standard C I/O functions are used and you can reset back to the standard C I/O functions by calling this function with the *fio* parameter set to NULL.

This function is useful for creating your own file system, such as storing all the bitmaps, fonts and icons that your application requires in a large file of your own format. This way end users browsing your program's data files will not be able to view any of the data (game developers may wish to keep the bitmaps used for levels in the game secret to make it harder for the user to cheat when playing the game).

This function allows you to overload the *fopen*, *fclose*, *fseek*, *ftell*, *fread* and *fwrite* functions used by MGL. See the *fileio_t* structure for more information.

MGL_setFontAntiAliasPalette

Sets the font anti-aliasing palette for color index modes.

Declaration

```
void MGLAPI MGL_setFontAntiAliasPalette(
    color_t colorfg,
    color_t color75,
    color_t color50,
    color_t color25,
    color_t colorbg)
```

Prototype In

mgraph.h

Parameters

<i>colorfg</i>	Color used to represent 100% of foreground
<i>color75</i>	Color used to represent 75% blend of foreground 25% background.
<i>color50</i>	Color used to represent 50% blend of foreground 50% background.
<i>color25</i>	Color used to represent 25% blend of foreground 75% background.
<i>colorbg</i>	Color used to represent 100% of background.

Description

This function sets the values of the anti-aliasing palette used to draw anti-aliased fonts when in non-blended mode in color index display modes. Anti-aliased fonts are rendered using a 5 level scheme, and the five colors passed in to this function set up the palette entries to be used for color index modes. It is up to the application program to ensure that the palette is set up correctly with the appropriate colors.

In RGB display modes, the colors are automatically computed from the foreground and background colors, so this function should not be called.

See Also

MGL_setFontBlendMode, *MGL_getFontAntiAliasPalette*

MGL_setFontBlendMode

Sets the blending mode for anti-aliased fonts.

Declaration

```
void MGLAPI MGL_setFontBlendMode(
    int type)
```

Prototype In

mgraph.h

Parameters

type Type of Blending mode, enumerated by *MGL_fontBlendType*.

Description

Sets the blending mode for anti-aliased fonts. The default blending mode of MGL_AA_NORMAL does not combine source and destination pixels, but chooses the colors from the anti-aliasing palette in 256 color modes, or from value blended between the current foreground and background colors.

If the font blending mode is set to MGL_AA_RGBBLEND, then all anti-aliased pixels in the font are blended both between the current foreground and background colors, and the destination pixels on the screen.

Using MGL_AA_NORMAL provides the fastest font anti-aliasing support, but it only works properly if you are drawing text over a solid background (such as text in a word processor type application). Using MGL_AA_RGBBLEND provides the highest quality anti-aliasing, and will correctly blend fonts over the top of existing images in the framebuffer.

Note: *MGL_AA_RGBBLEND is only available in modes with > 8 bits per pixel. Blending cannot be done in 8 bits per pixel display modes.*

See Also

MGL_getFontBlendMode, MGL_setFontAntiAliasPalette

MGL_setGammaRamp

Sets the hardware gamma correction ramp.

Declaration

```
ibool MGLAPI MGL_setGammaRamp(
    MGLDC *dc,
    palette_ext_t *gamma,
    int num,
    int index,
    ibool waitVRT)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to set gamma ramp values for
<i>gamma</i>	Pointer to array of gamma ramp values to set
<i>num</i>	Number of gamma entries to set
<i>index</i>	Starting index of first gamma entry to set
<i>waitVRT</i>	True if routine should sync to vertical retrace, false if not.

Description

This function sets part or all of the hardware gamma correction ramp for a device context. You can specify only a subset of the gamma ramp entries to be modified with the `startIndex` and `numColors` arguments. The changes to the gamma ramp take place immediately.

Note: *This function is only valid for RGB display modes, and will do nothing in color index modes.*

See Also

MGL_getGammaRamp

MGL_setLineStipple

Set the current line stipple pattern.

Declaration

```
void MGLAPI MGL_setLineStipple(
    ushort stipple)
```

Prototype In

mgraph.h

Parameters

stipple New 16 - bit stipple pattern to set.

Description

Sets the current line stipple pattern. The line stipple pattern is used to determine which pixels in the line get drawn depending on which bits in the pattern are set. The stipple pattern is a 16-bit value, and everywhere that a bit is set to a 1 in the pattern, a pixel will be drawn in the line. Everywhere that a bit is a 0, the pixel will be skipped in the line. Note that bit 0 in the stipple pattern corresponds to pixel 0,16,32,... in the line, bit 1 is pixel 1,17,33 etc. To create a line that is drawn as a 'dot dot dash dash' you would use the following value:

```
0011100111001001b or 0x39C9
```

Note that to enable stippled line mode you must call *MGL_setLineStyle*, with the *MGL_LINE_STIPPLE* parameter. Also note that stippled lines can only be 1 pixel wide, and the pen size will be ignored when drawing a stippled line.

See Also

MGL_setLineStyle, *MGL_setLineStippleCount*, *MGL_getLineStipple*

MGL_setLineStippleCount

Sets the current line stipple counter.

Declaration

```
void MGLAPI MGL_setLineStippleCount(  
    uint stippleCount)
```

Prototype In

mgraph.h

Parameters

stippleCount New line stipple counter to use

Description

Sets the current line stipple counter to a specific value. The line stipple counter is used to count the number of pixels that have been drawn in the line, and is updated after the line has been drawn. The purpose of this counter is to allow you to draw connected lines using a stipple pattern and the stippling will be continuous across the break in the lines. You can use this function to reset the stipple counter to a known value before drawing lines to force the stipple pattern to start at a specific bit position (usually resetting it to 0 before drawing a group of lines is sufficient).

See Also

MGL_setLineStyle, MGL_setLineStipple, MGL_getLineStippleCount

MGL_setLineStyle

Sets the current line style.

Declaration

```
void MGLAPI MGL_setLineStyle(
    int style)
```

Prototype In

mgraph.h

Parameters

style New line style to use

Description

Sets the current line style. MGL supports two different line styles, either pen style patterned lines (MGL_LINE_PENSTYLE) or stippled lines (MGL_LINE_STIPPLE). Pen style patterned lines are similar to those provided by QuickDraw for the Macintosh where lines are drawing using a rectangular pen that can have an arbitrary size and can be filled with an arbitrary pattern. Pen style patterned lines are the default. Stippled lines are similar to those used by CAD programs on the PC, and are 1-pixel wide lines that can be drawn using a 16-bit stipple mask. Stippled lines can be drawn very fast in hardware using the VBE/AF accelerator drivers.

In stippled line mode the line stipple pattern is used to determine which pixels in the line get drawn depending on which bits in the pattern are set. The stipple pattern is a 16-bit value, and everywhere that a bit is set to a 1 in the pattern, a pixel will be drawn in the line. Everywhere that a bit is a 0, the pixel will be skipped in the line. Note that bit 0 in the stipple pattern corresponds to pixel 0,16,32,... in the line, bit 1 is pixel 1,17,33 etc. To create a line that is drawn as a 'dot dot dash dash' you would use the following value:

```
0011100111001001b or 0x39C9
```

In stippled line mode the line stipple counter is used to count the number of pixels that have been drawn in the line, and is updated after the line has been drawn. The purpose of this counter is to allow you to draw connected lines using a stipple pattern and the stippling will be continuous across the break in the lines. You can use this function to reset the stipple counter to a known value before drawing lines to force the stipple pattern to start at a specific bit position (usually resetting it to 0 before drawing a group of lines is sufficient).

Note that VBE/AF 1.0 accelerated devices do not support the line stipple counter, so this counter is essentially reset to 0 every time that a line is drawn using the hardware. VBE/AF 2.0 will rectify this problem in the future.

See Also

MGL_setLineStipple, MGL_setLineStippleCount, MGL_getLineStyle

MGL_setOpenGLFuncs

Set the OpenGL rendering functions table pointer

Declaration

```
void GLAPIENTRY MGL_setOpenGLFuncs (  
    GLS_glFuncs *glFuncs,  
    GLS_gluFuncs *gluFuncs)
```

Prototype In

GL/gl.h

Parameters

<i>glFuncs</i>	Pointer to GLS_glFuncs structure to fill in
<i>gluFuncs</i>	Pointer to GLS_gluFuncs structure to fill in

Description

This function is called by the user application if the MGL libraries are stored in a DLL. By letting the MGL know about the OpenGL function pointer table, when the OpenGL implementation is swapped by the MGL it automatically updates the table in the user DLL to point to the newly loaded OpenGL entry points. This allows the code in the DLL to run with maximum performance for calls to OpenGL via function pointers.

MGL_setPalette

Sets the palette values for a device context.

Declaration

```
void MGLAPI MGL_setPalette(
    MGLDC *dc,
    palette_t *pal,
    int numColors,
    int startIndex)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to set palette values for
<i>pal</i>	Pointer to array of palette values to set
<i>numColors</i>	Number of color values to set
<i>startIndex</i>	Starting index of first color value to set

Description

This function sets part or all of the color palette for the device context. You can specify only a subset of the palette values to be modified with the `startIndex` and `numColors` arguments. Thus:

```
MGL_setPalette(dc, pal, 10, 50);
```

will program the 10 color indices from 50-60 with the values stored in the palette buffer 'pal'.

Note: *This routine does not actually change the value of the hardware palette. If you wish to change the hardware palette to reflect the new values, you will need to call the MGL_realizePalette function to update the hardware palette.*

Note: *You must ensure that you do not attempt to program invalid color indices! Use MGL_maxColor() to find the largest color index in color index modes.*

Note: *This function is also valid for RGB device contexts, and will simply set the color translation tables for these devices (used for drawing color index bitmaps and translating color index color values to RGB values).*

See Also

MGL_getPalette, MGL_setPaletteEntry, MGL_realizePalette

MGL_setPaletteEntry

Sets a single palette entry.

Declaration

```
void MGLAPI MGL_setPaletteEntry(
    MGLDC *dc,
    int entry,
    uchar red,
    uchar green,
    uchar blue)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to set palette entry in
<i>entry</i>	Palette index to program
<i>red</i>	Red component for palette entry
<i>green</i>	Green component for palette entry
<i>blue</i>	Blue component for palette entry

Description

Sets the color values of a single palette entry. If you wish to set more than a single palette index you should use the

MGL_setPalette routine which is faster for multiple entries. Note that this routine does not actually change the value of the hardware palette, and if you wish to change the hardware palette to reflect the new values, you will need to call the *MGL_realizePalette* function to update the hardware palette.

This function is also valid for RGB device contexts, and will simply set the color translation tables for these devices (used for drawing color index bitmaps and translating color index color values to RGB values).

See Also

MGL_getPaletteEntry, *MGL_setPalette*, *MGL_getPalette*, *MGL_realizePalette*

MGL_setPaletteSnowLevel

Sets the current palette snow level for a display device context.

Declaration

```
void MGLAPI MGL_setPaletteSnowLevel(  
    MGLDC *dc,  
    int level)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Display device context to set snow level for
<i>level</i>	New snow level to set

Description

This function sets the number of palette entries that can be programmed during a single vertical retrace before the onset of snow. By default MGL programs all 256 entries per retrace, but you may need to slow this down on systems with slower hardware that causes snow during multiple palette realization commands.

See Also

MGL_getPaletteSnowLevel, *MGL_setPalette*

MGL_setPenBitmapPattern

Downloads a new bitmap pattern into the driver.

Declaration

```
void MGLAPI MGL_setPenBitmapPattern(
    int index,
    const pattern_t *pat)
```

Prototype In

mgraph.h

Parameters

<i>index</i>	Index of the bitmap pattern slot to download into
<i>pat</i>	New bitmap pattern to download

Description

This function downloads a new bitmap pattern into the hardware on the graphics device. The MGL supports 8 patterns cached in the driver, allowing you to select one of them to be active at a time using the *MGL_usePenBitmapPattern* function. Bitmap patterns are used when rendering patterned primitives in the MGL_BITMAP_TRANSPARENT and MGL_BITMAP_OPAQUE pen styles. A bitmap pattern is defined as an 8 x 8 array of monochrome pixels, stored as an array of 8 bytes.

When filling in the MGL_BITMAP_TRANSPARENT mode, the foreground color is used to fill in all pixels in the bitmap pattern that are a 1. Where the pixels in the bitmap pattern are a 0, the original background color is retained. In the MGL_BITMAP_OPAQUE mode, the background color is used to fill in the pixels in the bitmap that are set to a 0.

Note: *After calling MGL_setPenBitmapPattern, the pattern will not become active until **after** you also call MGL_usePenBitmapPattern. This function only downloads the pattern into the pattern cache, and does not actually select the pattern for use (even if the old selected pattern was the same index).*

See Also

MGL_usePenBitmapPattern, MGL_getPenBitmapPattern, MGL_setPenPixmapPattern, MGL_setPenStyle

MGL_setPenPixmapPattern

Downloads a new pixmap pattern into the driver.

Declaration

```
void MGLAPI MGL_setPenPixmapPattern(
    int index,
    const pixpattern_t *pat)
```

Prototype In

mgraph.h

Parameters

<i>index</i>	Index of the pixmap pattern slot to download into
<i>pat</i>	New pixmap pattern to download

Description

This function downloads a new pixmap pattern into the hardware on the graphics device. The MGL supports 8 patterns cached in the driver, allowing you to select one of them to be active at a time using the *MGL_usePenPixmapPattern* function. Pixmap patterns are used when rendering patterned primitives in the MGL_PIXMAP pen style. A pixmap pattern is defined as an 8 x 8 array of color pixels, stored differently depending on the color depth for the pattern. When filling in MGL_PIXMAP mode, the foreground and background color values are not used, and the pixel colors are obtained directly from the pixmap pattern colors.

Note: *After calling MGL_setPenPixmapPattern, the pattern will not become active until **after** you also call MGL_usePenPixmapPattern. This function only downloads the pattern into the pattern cache, and does not actually select the pattern for use (even if the old selected pattern was the same index).*

See Also

MGL_usePenPixmapPattern, MGL_getPenPixmapPattern, MGL_setPenBitmapPattern, MGL_setPenStyle

MGL_setPenPixmapTransparent

Sets the currently active pixmap pattern transparent color.

Declaration

```
void MGLAPI MGL_setPenPixmapTransparent(  
    color_t color)
```

Prototype In

mgraph.h

Parameters

color New transparent color for pixmap patterns

Description

This function sets the currently active pixmap pattern transparent color. This is used to determine which pixels in the pixmap pattern are transparent when the device context has the MGL_PIXMAP_TRANSPARENT pen style active.

See Also

MGL_setPenPixmapPattern, MGL_getPenPixmapTransparent

MGL_setPenSize

Sets the current pen size.

Declaration

```
void MGLAPI MGL_setPenSize(  
    int h,  
    int w)
```

Prototype In

mgraph.h

Parameters

<i>h</i>	Height of the pen in pixels
<i>w</i>	Width of the pen in pixels

Description

Sets the size of the current pen size in pixels. The default pen is 1 pixel by 1 pixel in dimensions, however you can change this to whatever value you like. When primitives are rasterized with a pen other than the default, the pixels in the pen always lie to the right and below the current pen position.

See Also

MGL_getPenSize

MGL_setPenStyle

Sets the current pen style.

Declaration

```
void MGLAPI MGL_setPenStyle(  
    int style)
```

Prototype In

mgraph.h

Parameters

style New pen style to use

Description

Returns the currently active pen style. Pen styles supported by the SciTech MGL are enumerated in *MGL_penStyleType*.

When filling in the MGL_BITMAP_TRANSPARENT mode, the foreground color is used to fill in all pixels in the bitmap pattern that are a 1. Where the pixels in the bitmap pattern are a 0, the original background color is retained. In the MGL_BITMAP_OPAQUE mode, the background color is used to fill in the pixels in the bitmap that are set to a 0. When filling in MGL_PIXMAP mode, the foreground and background color values are not used, and the pixel colors are obtained directly from the pixmap pattern colors.

See Also

MGL_getPenStyle, *MGL_setPenBitmapPattern*

MGL_setPlaneMask

Sets the current plane mask for all rendering operations.

Declaration

```
void MGLAPI MGL_setPlaneMask(  
    ulong mask)
```

Prototype In

mgraph.h

Parameters

mask New plane mask to make active

Description

This function sets the current plane mask for all rendering operations, 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: *A plane mask of 0xFFFFFFFF will disable plane masking operation, and this is the default plane mask enabled by the MGL when it starts up.*

Note: *Not all hardware supports plane masking in hardware, and if hardware plane masking is not available, enabling this will cause the rendering to be done entirely in software.*

See Also

MGL_getPlaneMask

MGL_setPolygonType

Sets the current polygon type.

Declaration

```
void MGLAPI MGL_setPolygonType(  
    int type)
```

Prototype In

mgraph.h

Parameters

type New polygon type

Description

Sets the current polygon type. You can change this value to force MGL to work with a specific polygon type (and to avoid the default automatic polygon type checking). Polygon types supported by the SciTech MGL are enumerated in *MGL_polygonType*.

If you expect to be drawing lots of complex or convex polygons, setting the polygon type can result in faster polygon rasterizing. Note that this setting does not affect the specialized triangle and quadrilateral rasterizing routines.

See Also

MGL_getPolygonType, *MGL_fillPolygon*

MGL_setRelViewport

Sets a viewport relative to the current viewport.

Declaration

```
void MGLAPI MGL_setRelViewport(  
    rect_t view)
```

Prototype In

mgraph.h

Parameters

view Bounding rectangle for the new viewport

Description

Sets the current viewport to the viewport specified by *view*, relative to the currently active viewport. The new viewport is restricted to fall within the bounds of the currently active viewport. Note that when the viewport is changing, the viewport origin is always reset back to (0,0).

All output in MGL is relative to the current viewport, so by changing the viewport to a new value you can make all output appear in a different rectangular portion of the video display.

See Also

MGL_getViewport, *MGL_setViewport*, *MGL_clearViewport*, *MGL_setClipRect*,
MGL_setViewportOrg

MGL_setRelViewportDC

Sets a viewport relative to the current viewport for a specific DC.

Declaration

```
void MGLAPI MGL_setRelViewportDC(  
    MGLDC *dc,  
    rect_t view)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to change viewport for
<i>view</i>	Bounding rectangle for the new viewport

Description

This function is the same as *MGL_setRelViewport*, however the device context does not have to be the current device context.

See Also

MGL_setRelViewport, *MGL_getViewport*, *MGL_setViewport*, *MGL_clearViewport*,
MGL_setClipRect, *MGL_setViewportOrg*

MGL_setResult

Sets the internal MGL result flag.

Declaration

```
void MGLAPI MGL_setResult(  
    int result)
```

Prototype In

mgraph.h

Parameters

result New internal result flag

Description

Sets the internal MGL result flag to the specified value. This routine is primarily for extension libraries, but you can use it to add your own extension functions to the MGL that will return result codes in the same manner as the MGL itself;

See Also

MGL_result, *MGL_errorMsg*

MGL_setSpaceExtra

Sets the current space extra value.

Declaration

```
void MGLAPI MGL_setSpaceExtra(  
    int extra)
```

Prototype In

mgraph.h

Parameters

extra New space extra value

Description

Sets the current space extra value used when drawing text in the current font. The space extra value is normally zero, but can be a positive or negative value. When this value is positive, it will insert extra space between the characters in a font and making this value negative will make the characters run on top of each other.

See Also

MGL_getSpaceExtra, *MGL_drawStr*

MGL_setSuspendAppCallback

Sets the fullscreen suspend application callback function.

Declaration

```
void MGLAPI MGL_setSuspendAppCallback(
    MGL_suspend_cb_t saveState)
typedef int (MGLAPI MGL_suspend_cb_t) (MGLDC *dc, int flags)
```

Prototype In

mgraph.h

Parameters

saveState New suspend app callback to be used.

Description

This function is used to register an application suspend callback function. This is used in fullscreen modes under Windows and is called by MGL when the application's fullscreen window has lost the input focus and the system has returned to the normal GUI desktop. The focus can be lost due to the user hitting a System Key combination such as Alt-Tab or Ctrl-Esc which forces your fullscreen application into the background. The MGL takes care of all the important details such as saving and restoring the state of the hardware, so all your suspend application callback needs to do is save the current state of your program so that when the request is made to re-activate your application, you can redraw the screen and continue from where you left off.

When the MGL detects that your application has been suspended it will call the registered callback with a combination of the following flags:

<i>Flag</i>	<i>Meaning</i>
<i>MGL_DEACTIVATE</i>	This flag will be sent when your application has lost the input focus and has been suspended.
<i>MGL_REACTIVATE</i>	This flag will be sent when the user re-activates your fullscreen application again indicating that the fullscreen mode has now been restored and the application must redraw the display ready to continue on.

By default if you have not installed a suspend callback handler, the MGL will simply restore the display to the original state with the screen cleared to black when the application is re-activated. If your application is a game or animation that is continuously updating the screen, you won't need to do anything as the next frame in the animation will re-draw the screen correctly.

If your application is caching bitmaps in offscreen video memory however, all of the bitmaps will need to be restored to the offscreen display device context when the application is restored (the offscreen memory will be cleared to black also).

Note: *By the time your callback is called, the display memory may have already been lost under DirectDraw. Hence you cannot save and restore the contents of the display memory, but must be prepared to redraw the entire display when the callback is called with the MGL_REACTIVATE flag set.*

MGL_setTextDirection

Sets the current text direction.

Declaration

```
void MGLAPI MGL_setTextDirection(  
    int direction)
```

Prototype In

mgraph.h

Parameters

direction New text direction value

Description

Sets the current text direction. Directions supported by the SciTech MGL are enumerated in *MGL_textJustType*.

See Also

MGL_getTextDirection, *MGL_drawStr*

MGL_setTextEncoding

Changes current text encoding.

Declaration

```
void MGLAPI MGL_setTextEncoding(  
    int encoding)
```

Prototype In

mgraph.h

Parameters

encoding text encoding, one of *MGL_textEncodingType* constants

Description

This function changes current text encoding (also referred to as charset). Encoding is used by *MGL_drawStr* and *MGL_drawStrXY* to interpret 8bit input string and translate it to wide char Unicode representation.

Default encoding is *MGL_ENCODING_ASCII* that only guarantees that characters with ASCII value <128 will be rendered correctly.

Encodings only work with TrueType fonts. Calling this function has no effect on *MGL_drawStr_W* and *MGL_drawStrXY_W*.

See Also

MGL_drawStr, *MGL_drawStrXY*, *MGL_textEncodingType*

MGL_setTextJustify

Sets the current text horizontal and vertical justification.

Declaration

```
void MGLAPI MGL_setTextJustify(  
    int horiz,  
    int vert)
```

Prototype In

mgraph.h

Parameters

<i>horiz</i>	New horizontal text justification value
<i>vert</i>	New vertical text justification value

Description

Sets the current text justification values. Horizontal and vertical justification type supported by the SciTech MGL are enumerated in *MGL_textJustType*.

See Also

MGL_getTextJustify

MGL_setTextSettings

Restores the current text settings.

Declaration

```
void MGLAPI MGL_setTextSettings(  
    text_settings_t *settings)
```

Prototype In

mgraph.h

Parameters

settings Text settings to restore

Description

Restores a set of previously saved text settings. This routine provides a way to save and restore all the values relating to the rasterizing of text in MGL with a single function call.

See Also

MGL_getTextSettings

MGL_setTextSize

Sets the current text scaling factors

Declaration

```
void MGLAPI MGL_setTextSize(
    int numerx,
    int denomx,
    int numery,
    int denomy)
```

Prototype In

mgraph.h

Parameters

<i>numerx</i>	x scaling numerator value
<i>denomx</i>	x scaling denominator value
<i>numery</i>	y scaling numerator value
<i>denomy</i>	y scaling denominator value

Description

Sets the current text scaling factors used by MGL. The text size values define an integer scaling factor to be used, where the actual values will be computed using the following formula:

$$\text{scaled} = \frac{\text{unscaled} * \text{numer}}{\text{denom}}$$

Note: *MGL can only scale vectored fonts.*

See Also

MGL_getTextSize

MGL_setViewport

Sets the currently active viewport.

Declaration

```
void MGLAPI MGL_setViewport(  
    rect_t view)
```

Prototype In

mgraph.h

Parameters

view New global viewport bounding rectangle

Description

Sets the dimensions of the currently active viewport. These dimensions are global to the entire display area used by the currently active video device driver. Note that when the viewport is changing, the viewport origin and current position is always reset back to (0,0).

All output in MGL is relative to the current viewport, so by changing the viewport to a new value you can make all output appear in a different rectangular portion of the video display.

Note: *Setting the viewport also implicitly sets the clip rectangle to cover the entire viewport bounds, which also clears any user defined complex clip region.*

See Also

MGL_getViewport, MGL_setRelViewport, MGL_clearViewport, MGL_setClipRect, MGL_setViewportOrg

MGL_setViewportDC

Sets the currently active viewport for a specific DC.

Declaration

```
void MGLAPI MGL_setViewportDC(  
    MGLDC *dc,  
    rect_t view)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to change viewport for
<i>view</i>	New global viewport bounding rectangle

Description

This function is the same as *MGL_setViewport*, however the device context does not have to be the current device context.

See Also

MGL_setViewport, *MGL_getViewport*, *MGL_setRelViewport*, *MGL_clearViewport*, *MGL_setClipRect*, *MGL_setViewportOrg*

MGL_setViewportOrg

Sets the logical viewport origin.

Declaration

```
void MGLAPI MGL_setViewportOrg(  
    point_t org)
```

Prototype In

mgraph.h

Parameters

org New logical viewport origin.

Description

This function sets the currently active viewport origin. When a new viewport is set with the *MGL_setViewport* function, the viewport origin is reset to (0,0), which means that any primitives drawn at pixel location (0,0) will appear at the top left hand corner of the viewport.

You can change the logical coordinate of the viewport origin to any value you please, which will effectively offset all drawing within the currently active viewport. Hence if you set the viewport origin to (10,10), drawing a pixel at (10,10) would make it appear at the top left hand corner of the viewport.

See Also

MGL_setViewportOrgDC, *MGL_getViewportOrg*, *MGL_setViewport*

MGL_setViewportOrgDC

Sets the logical viewport origin for a specific DC.

Declaration

```
void MGLAPI MGL_setViewportOrgDC(  
    MGLDC *dc,  
    point_t org)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to change viewport for
<i>org</i>	New logical viewport origin.

Description

This function is the same as *MGL_setViewportOrg*, however the device context does not have to be the current device context.

See Also

MGL_setViewportOrg, *MGL_getViewportOrg*, *MGL_setViewport*

MGL_setVisualPage

Sets the currently visible hardware video page for a display device context.

Declaration

```
void MGLAPI MGL_setVisualPage(
    MGLDC *dc,
    int page,
    int waitVRT)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Display device context
<i>page</i>	New hardware display page
<i>waitVRT</i>	Wait for retrace flag (<i>MGL_waitVRTFlagType</i>)

Description

This function sets the currently visible hardware video page for a display device context. The first hardware video page is number 0, the second is 1 and so on. The number of available hardware video pages depends on the type of underlying hardware, the video mode resolution and amount of video memory installed. Thus not all video modes support multiple hardware video pages.

When the visible display page is changed, you should normally allow MGL to sync to the vertical retrace by passing a value of *MGL_waitVRT* in the *waitVRT* parameter to ensure that the change occurs in the correct place, and that you don't get flicker effects on the display. If you have more than two display pages for your display device context, you can also pass in a value of *MGL_tripleBuffer*, and if the hardware supports triple buffering the MGL will implement proper support for triple buffering. If the hardware does not support triple buffering, *MGL_tripleBuffer* behaves the same as *MGL_waitVRT*. Triple buffering is a mechanism that allows the MGL to return immediately without waiting for the vertical sync period after changing the visual display page, but ensure that if the application runs at a frame rate higher than the refresh rate of the graphics adapter (ie: 60fps if the refresh is 60Hz), the frame rate will lock to the vertical refresh frequency and you will not get any flicker.

You may however want to turn off the vertical retrace synching if you are synching up with the retrace period using some other means by passing a value of *MGL_dontWait* to the *waitVRT* parameter. This is also useful if you are measuring the performance of your application and you want it to run at full speed without the overhead of waiting for the vertical retrace.

Note that if you wish to implement both double buffering and hardware scrolling or panning, you should call the *MGL_setDisplayStart* function first with *waitVRT* set to -1, and then call this function with *waitVRT* set to *MGL_waitVRT* or *MGL_tripleBuffer* to actually update the hardware. The first call to *MGL_setDisplayStart* simply updates the

internal display start variables but does not program the hardware. For more information please see the *MGL_setDisplayStart* function.

See Also

MGL_getVisualPage, *MGL_getActivePage*, *MGL_setActivePage*, *MGL_swapBuffers*, *MGL_setDisplayStart*

MGL_setWriteMode

Sets the current write mode operation.

Declaration

```
void MGLAPI MGL_setWriteMode(  
    int mode)
```

Prototype In

mgraph.h

Parameters

mode New write mode operation to use

Description

Sets the currently active write mode. Write mode operations supported by the SciTech MGL for all output primitives are enumerated in *MGL_writeModeType*.

See Also

MGL_getWriteMode

MGL_singleBuffer

Returns the display device context single buffered mode.

Declaration

```
void MGLAPI MGL_singleBuffer(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Display device context

Description

This function puts the display device context into single buffer mode. The active display page is made to be the same as the current visual display page for hardware double buffering. This may or may not be the first hardware video page.

See Also

MGL_doubleBuffer, MGL_swapBuffers

MGL_size_x

Returns the total device x coordinate dimensions.

Declaration

```
int MGLAPI MGL_size_x(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Display device context

Return Value

Number of pixels in x direction for entire device - 1

Description

Returns the total number of pixels available along the x coordinate axis for the currently active device context. This is different than the *MGL_maxx* routine which returns the dimensions of the currently active viewport.

See Also

MGL_size_y, *MGL_maxx*, *MGL_maxy*

MGL_sizey

Returns the total device y coordinate dimensions.

Declaration

```
int MGLAPI MGL_sizey(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Display device context

Return Value

Number of pixels in y direction for entire device - 1

Description

Returns the total number of pixels available along the y coordinate axis for the currently active device context. This is different than the *MGL_maxy* routine which returns the dimensions of the currently active viewport.

See Also

MGL_sizex, *MGL_maxx*

MGL_srand

Reseed MGL random number generator.

Declaration

```
void MGLAPI MGL_srand(  
    uint seed)
```

Prototype In

mgraph.h

Parameters

seed New seed value for the random number generator

Description

This function reseeds the random number generator to start generating a new sequence of numbers. Generally this function is used to randomize the generator by seeding it with the value obtained from the MGL_getTicks function.

See Also

MGL_random, MGL_randoml

MGL_srcTransBlit

Copies a block of image data with source transparency.

Declaration

```
void MGL_srcTransBlit(
    MGLDC *dst,
    MGLDC *src,
    rect_t srcRect,
    int dstLeft,
    int dstTop,
    color_t transparent,
    int op)
```

Prototype In

mgraph.h

Parameters

<i>dst</i>	Destination device context
<i>src</i>	Source device context
<i>srcRect</i>	Rectangle defining source image
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>transparent</i>	Transparent color to skip in source image
<i>op</i>	Write mode to use during Blt

Description

This function is the same as *MGL_srcTransBlitCoord*, however it takes a rectangle as a parameter instead of the four coordinates of a rectangle.

See Also

MGL_bitBlt, *MGL_bitBltCoord*, *MGL_srcTransBlit*, *MGL_srcTransBlitCoord*,
MGL_dstTransBlit, *MGL_dstTransBlitCoord*, *MGL_bitBltPatt*, *MGL_bitBltPattCoord*,
MGL_bitBltFx, *MGL_bitBltFxCoord*, *MGL_stretchBlt*, *MGL_stretchBltCoord*,
MGL_stretchBltFx, *MGL_stretchBltFxCoord*, *MGL_copyPage*, *MGL_copyPageCoord*

MGL_srcTransBlitCoord

Copies a block of image data with source transparency.

Declaration

```
void MGLAPI MGL_srcTransBlitCoord(
    MGLDC *dst,
    MGLDC *src,
    int left,
    int top,
    int right,
    int bottom,
    int dstLeft,
    int dstTop,
    color_t transparent,
    int op)
```

Prototype In

mgraph.h

Parameters

<i>dst</i>	Destination device context
<i>src</i>	Source device context
<i>left</i>	Left coordinate of source image
<i>top</i>	Top coordinate of source image
<i>right</i>	Right coordinate of source image
<i>bottom</i>	Bottom coordinate of source image
<i>dstLeft</i>	Left coordinate of destination
<i>dstTop</i>	Top coordinate of destination
<i>transparent</i>	Transparent color to skip in source image
<i>op</i>	Write mode to use during Blt

Description

Copies a block of bitmap data from one device context to another with either source or destination transparency. When transferring the data with source transparency, for pixels in the source image that are equal to the specified transparent color, the related pixel in the destination buffer will remain untouched. This allows you to quickly transfer sprites between device contexts with a single color being allocated as a transparent color.

This routine has been highly optimized for maximum performance in all pixel depths, so will provide a very fast method for performing transparent sprite animation. However you may find that if you can use alternative techniques to pre-compile the sprites (like using run length encoding etc.) you will be able to build faster software based sprite animation code that can directly access the device context surface. However this routine can also be used to perform hardware accelerated Blt's between offscreen memory device's and the display device when running in fullscreen modes, providing the hardware accelerator (if present) can support this operation. If you have a hardware

accelerator capable of this, this will provide the ultimate performance for transparent sprite animation.

The source and destination rectangles are clipped according to the current clipping rectangles for the source and destination device contexts respectively.

Note: *If you are doing pixel format conversion at the same time (ie: color depth for source bitmap is different to the destination bitmap), then the transparent color value must be set to the traslated destination pixel format. Ie: if you are blitting an 8bpp bitmap to a 32bpp device context, the transparent color must be a 32bpp value.*

Note: *This routine also only works with pixel depths that are at least 4 bits deep.*

See Also

MGL_bitBlit, MGL_bitBlitCoord, MGL_srcTransBlit, MGL_srcTransBlitCoord, MGL_dstTransBlit, MGL_dstTransBlitCoord, MGL_bitBlitPatt, MGL_bitBlitPattCoord, MGL_bitBlitFx, MGL_bitBlitFxCoord, MGL_stretchBlit, MGL_stretchBlitCoord, MGL_stretchBlitFx, MGL_stretchBlitFxCoord, MGL_copyPage, MGL_copyPageCoord

MGL_startStereo

Enables free running stereo display mode

Declaration

```
void MGLAPI MGL_startStereo(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Description

This function enables the free running stereo display mode for LC shutter glasses. This function only works if the display device context you created was a stereo display device context created with *MGL_createStereoDisplayDC*. By default free running stereo mode is on when you create the stereo display device context.

See Also

MGL_stopStereo, *MGL_createStereoDisplayDC*

MGL_stopStereo

Disables free running stereo display mode

Declaration

```
void MGLAPI MGL_stopStereo(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Description

This function disables the free running stereo display mode for LC shutter glasses. This function only works if the display device context you created was a stereo display device context created with *MGL_createStereoDisplayDC*. By default free running stereo mode is on when you create the stereo display device context, and you can use this function to disable it in parts of your application that don't require stereo (such as when navigating the menu system etc). Note that when stereo mode is disabled, the MGL always displays from the left eye buffer.

See Also

MGL_startStereo, *MGL_createStereoDisplayDC*

MGL_stretchBitmap

Stretches a lightweight bitmap to the specified rectangle.

Declaration

```
void MGLAPI MGL_stretchBitmap(
    MGLDC *dc,
    int dstLeft,
    int dstTop,
    int dstRight,
    int dstBottom,
    const bitmap_t *bitmap,
    int op)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to display bitmap on
<i>dstLeft</i>	Left coordinate to stretch bitmap to
<i>dstTop</i>	Top coordinate to stretch bitmap to
<i>dstRight</i>	Right coordinate to stretch bitmap to
<i>dstBottom</i>	Bottom coordinate to stretch bitmap to
<i>bitmap</i>	Bitmap to display
<i>op</i>	Write mode to use when drawing bitmap

Description

Stretches a lightweight bitmap to the specified destination rectangle on the device context. The bitmap can be in any color format, and will be translated as necessary to the color format required by the current device context.

When this function is called for 4 and 8 bit source bitmaps being copied to either 4 or 8bpp destination device contexts, MGL first checks if the color palettes for the source and destination bitmaps are the same. If they are not, MGL translates the pixel values from the source bitmap to the destination color palette, looking for the closest match color if an exact match is not found. In order to obtain maximum performance for blt'ing bitmaps in color index modes, you should ensure that the color palette in the source device matches the color palette in the destination device to avoid on the fly palette translation. If you know in advance that the palette is identical for a series of blit operations, you can turn off all identity palette checking in MGL with the *MGL_checkIdentityPalette* function.

When this function is called for 4 and 8 bit source bitmaps being copied to RGB destination device contexts, MGL will convert the pixels in the source bitmap using the source bitmap palette to map them to the destination pixel format. If however you know in advance that the palette for all source bitmaps is identical for a series of blit operations, you can use the *MGL_checkIdentityPalette* function to disable source palette translation. In this case the MGL will translate all color index bitmaps using the pre-

defined color translation palette stored in the destination device context. You would then set the destination device context palette to the common palette for all blit operations using *MGL_setPalette*. If you are translating a lot of color index bitmaps, this will increase performance by avoiding the need to convert the palette entries to the destination pixel format for every blit operation.

Note also that if the source bitmap palette pointer is set to NULL, palette translation is automatically avoided (ie: has the effect of forcing *MGL_checkIdentityPalette* to false just for that bitmap).

Supported write modes are enumerated in *MGL_writeModeType*.

See Also

MGL_loadBitmap, *MGL_putBitmap*, *MGL_putBitmapSection*, *MGL_putBitmapSrcTrans*,
MGL_putBitmapSrcTransSection, *MGL_putBitmapDstTrans*,
MGL_putBitmapDstTransSection, *MGL_putBitmapMask*, *MGL_putBitmapPatt*,
MGL_putBitmapPattSection, *MGL_putBitmapFx*, *MGL_putBitmapFxSection*,
MGL_stretchBitmap, *MGL_stretchBitmapSection*, *MGL_stretchBitmapFx*,
MGL_stretchBitmapFxSection, *MGL_putIcon*

MGL_stretchBitmapFx

Stretches a lightweight bitmap to the specified rectangle, while applying different effects in the process.

Declaration

```
void MGLAPI MGL_stretchBitmapFx(
    MGLDC *dc,
    int dstLeft,
    int dstTop,
    int dstRight,
    int dstBottom,
    const bitmap_t *bitmap,
    bltfx_t *fx)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to display bitmap on
<i>dstLeft</i>	Left coordinate to stretch bitmap to
<i>dstTop</i>	Top coordinate to stretch bitmap to
<i>dstRight</i>	Right coordinate to stretch bitmap to
<i>dstBottom</i>	Bottom coordinate to stretch bitmap to
<i>bitmap</i>	Bitmap to display
<i>fx</i>	Information describing the effects to apply to the blit (<i>bltfx_t</i>)

Description

Stretches a lightweight bitmap to the specified destination rectangle on the device context, while applying optional effects. This function behaves identically to the *MGL_putBitmapFx* function, except it includes the ability to stretch the bitmap at the same time. You can use the flags member of the *bltfx_t* structure to define whether stretching is done using nearest color stretching or if pixel interpolation will be used. If you don't specify any flags, nearest color stretching will be used. The bitmap can be in any color format, and will be translated as necessary to the color format required by the current device context.

When this function is called for 4 and 8 bit source bitmaps being copied to either 4 or 8bpp destination device contexts, MGL first checks if the color palettes for the source and destination bitmaps are the same. If they are not, MGL translates the pixel values from the source bitmap to the destination color palette, looking for the closest match color if an exact match is not found. In order to obtain maximum performance for blt'ing bitmaps in color index modes, you should ensure that the color palette in the source device matches the color palette in the destination device to avoid on the fly palette translation. If you know in advance that the palette is identical for a series of blit operations, you can turn off all identity palette checking in MGL with the *MGL_checkIdentityPalette* function.

When this function is called for 4 and 8 bit source bitmaps being copied to RGB destination device contexts, MGL will convert the pixels in the source bitmap using the source bitmap palette to map them to the destination pixel format. If however you know in advance that the palette for all source bitmaps is identical for a series of blit operations, you can use the *MGL_checkIdentityPalette* function to disable source palette translation. In this case the MGL will translate all color index bitmaps using the pre-defined color translation palette stored in the destination device context. You would then set the destination device context palette to the common palette for all blit operations using *MGL_setPalette*. If you are translating a lot of color index bitmaps, this will increase performance by avoiding the need to convert the palette entries to the destination pixel format for every blit operation.

Note also that if the source bitmap palette pointer is set to NULL, palette translation is automatically avoided (ie: has the effect of forcing *MGL_checkIdentityPalette* to false just for that bitmap).

See Also

MGL_loadBitmap, *MGL_putBitmap*, *MGL_putBitmapSection*, *MGL_putBitmapSrcTrans*, *MGL_putBitmapSrcTransSection*, *MGL_putBitmapDstTrans*, *MGL_putBitmapDstTransSection*, *MGL_putBitmapMask*, *MGL_putBitmapPatt*, *MGL_putBitmapPattSection*, *MGL_putBitmapFx*, *MGL_putBitmapFxSection*, *MGL_stretchBitmap*, *MGL_stretchBitmapSection*, *MGL_stretchBitmapFx*, *MGL_stretchBitmapFxSection*, *MGL_putIcon*

MGL_stretchBitmapFxSection

Stretches a section of a lightweight bitmap to the specified device context, while applying different effects in the process.

Declaration

```
void MGLAPI MGL_stretchBitmapFxSection(
    MGLDC *dc,
    int left,
    int top,
    int right,
    int bottom,
    int dstLeft,
    int dstTop,
    int dstRight,
    int dstBottom,
    const bitmap_t *bitmap,
    bltfx_t *fx)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to display bitmap on
<i>left</i>	Left coordinate of section to stretch
<i>top</i>	Top coordinate of section to stretch
<i>right</i>	Right coordinate of section to stretch
<i>bottom</i>	Bottom coordinate of section to stretch
<i>dstLeft</i>	Left coordinate to stretch bitmap to
<i>dstTop</i>	Top coordinate to stretch bitmap to
<i>dstRight</i>	Right coordinate to stretch bitmap to
<i>dstBottom</i>	Bottom coordinate to stretch bitmap to
<i>bitmap</i>	Bitmap to display
<i>fx</i>	Information describing the effects to apply to the blit (<i>bltfx_t</i>)

Description

Stretches a section of a lightweight bitmap to the specified destination rectangle on the device context, while applying optional effects. This function behaves identically to the MGL_putBitmapSectionFx function, except it includes the ability to stretch the bitmap at the same time. You can use the flags member of the *bltfx_t* structure to define whether stretching is done using nearest color stretching or if pixel interpolation will be used. If you don't specify any flags, nearest color stretching will be used. The bitmap can be in any color format, and will be translated as necessary to the color format required by the current device context.

When this function is called for 4 and 8 bit source bitmaps being copied to either 4 or 8bpp destination device contexts, MGL first checks if the color palettes for the source and destination bitmaps are the same. If they are not, MGL translates the pixel values

from the source bitmap to the destination color palette, looking for the closest match color if an exact match is not found. In order to obtain maximum performance for blt'ing bitmaps in color index modes, you should ensure that the color palette in the source device matches the color palette in the destination device to avoid on the fly palette translation. If you know in advance that the palette is identical for a series of blit operations, you can turn off all identity palette checking in MGL with the *MGL_checkIdentityPalette* function.

When this function is called for 4 and 8 bit source bitmaps being copied to RGB destination device contexts, MGL will convert the pixels in the source bitmap using the source bitmap palette to map them to the destination pixel format. If however you know in advance that the palette for all source bitmaps is identical for a series of blit operations, you can use the *MGL_checkIdentityPalette* function to disable source palette translation. In this case the MGL will translate all color index bitmaps using the pre-defined color translation palette stored in the destination device context. You would then set the destination device context palette to the common palette for all blit operations using *MGL_setPalette*. If you are translating a lot of color index bitmaps, this will increase performance by avoiding the need to convert the palette entries to the destination pixel format for every blit operation.

Note also that if the source bitmap palette pointer is set to NULL, palette translation is automatically avoided (ie: has the effect of forcing *MGL_checkIdentityPalette* to false just for that bitmap).

See Also

MGL_loadBitmap, *MGL_putBitmap*, *MGL_putBitmapSection*, *MGL_putBitmapSrcTrans*, *MGL_putBitmapSrcTransSection*, *MGL_putBitmapDstTrans*, *MGL_putBitmapDstTransSection*, *MGL_putBitmapMask*, *MGL_putBitmapPatt*, *MGL_putBitmapPattSection*, *MGL_putBitmapFx*, *MGL_putBitmapFxSection*, *MGL_stretchBitmap*, *MGL_stretchBitmapSection*, *MGL_stretchBitmapFx*, *MGL_stretchBitmapFxSection*, *MGL_putIcon*

MGL_stretchBitmapSection

Stretches a section of a lightweight bitmap to the specified device context.

Declaration

```
void MGLAPI MGL_stretchBitmapSection(
    MGLDC *dc,
    int left,
    int top,
    int right,
    int bottom,
    int dstLeft,
    int dstTop,
    int dstRight,
    int dstBottom,
    const bitmap_t *bitmap,
    int op)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to display bitmap on
<i>left</i>	Left coordinate of section to stretch
<i>top</i>	Top coordinate of section to stretch
<i>right</i>	Right coordinate of section to stretch
<i>bottom</i>	Bottom coordinate of section to stretch
<i>dstLeft</i>	Left coordinate to stretch bitmap to
<i>dstTop</i>	Top coordinate to stretch bitmap to
<i>dstRight</i>	Right coordinate to stretch bitmap to
<i>dstBottom</i>	Bottom coordinate to stretch bitmap to
<i>bitmap</i>	Bitmap to display
<i>op</i>	Write mode to use when drawing bitmap

Description

Stretches a section of a lightweight bitmap to the specified destination rectangle on the device context. The bitmap can be in any color format, and will be translated as necessary to the color format required by the current device context.

When this function is called for 4 and 8 bit source bitmaps being copied to either 4 or 8bpp destination device contexts, MGL first checks if the color palettes for the source and destination bitmaps are the same. If they are not, MGL translates the pixel values from the source bitmap to the destination color palette, looking for the closest match color if an exact match is not found. In order to obtain maximum performance for blt'ing bitmaps in color index modes, you should ensure that the color palette in the source device matches the color palette in the destination device to avoid on the fly palette translation. If you know in advance that the palette is identical for a series of blit

operations, you can turn off all identity palette checking in MGL with the *MGL_checkIdentityPalette* function.

When this function is called for 4 and 8 bit source bitmaps being copied to RGB destination device contexts, MGL will convert the pixels in the source bitmap using the source bitmap palette to map them to the destination pixel format. If however you know in advance that the palette for all source bitmaps is identical for a series of blit operations, you can use the *MGL_checkIdentityPalette* function to disable source palette translation. In this case the MGL will translate all color index bitmaps using the pre-defined color translation palette stored in the destination device context. You would then set the destination device context palette to the common palette for all blit operations using *MGL_setPalette*. If you are translating a lot of color index bitmaps, this will increase performance by avoiding the need to convert the palette entries to the destination pixel format for every blit operation.

Note also that if the source bitmap palette pointer is set to NULL, palette translation is automatically avoided (ie: has the effect of forcing *MGL_checkIdentityPalette* to false just for that bitmap).

Supported write modes are enumerated in *MGL_writeModeType*.

See Also

MGL_loadBitmap, *MGL_putBitmap*, *MGL_putBitmapSection*, *MGL_putBitmapSrcTrans*,
MGL_putBitmapSrcTransSection, *MGL_putBitmapDstTrans*,
MGL_putBitmapDstTransSection, *MGL_putBitmapMask*, *MGL_putBitmapPatt*,
MGL_putBitmapPattSection, *MGL_putBitmapFx*, *MGL_putBitmapFxSection*,
MGL_stretchBitmap, *MGL_stretchBitmapSection*, *MGL_stretchBitmapFx*,
MGL_stretchBitmapFxSection, *MGL_putIcon*

MGL_stretchBlt

Stretches a block of image data from one device context to another.

Declaration

```
void MGL_stretchBlt(
    MGLDC dst,
    MGLDC src,
    rect_t srcRect,
    rect_t dstRect,
    int op)
```

Prototype In

mgraph.h

Parameters

<i>dst</i>	Destination device context
<i>src</i>	Source device context
<i>srcRect</i>	Rectangle defining source image
<i>dstRect</i>	Rectangle defining destination image
<i>op</i>	Write mode to use during Blt

Description

This function is the same as *MGL_stretchBltCoord*, however it takes entire rectangles as arguments instead of coordinates.

See Also

MGL_bitBlt, *MGL_bitBltCoord*, *MGL_srcTransBlt*, *MGL_srcTransBltCoord*,
MGL_dstTransBlt, *MGL_dstTransBltCoord*, *MGL_bitBltPatt*, *MGL_bitBltPattCoord*,
MGL_bitBltFx, *MGL_bitBltFxCoord*, *MGL_stretchBlt*, *MGL_stretchBltCoord*,
MGL_stretchBltFx, *MGL_stretchBltFxCoord*, *MGL_copyPage*, *MGL_copyPageCoord*

MGL_stretchBltCoord

Stretches a block of image data from one device context to another.

Declaration

```
void MGLAPI MGL_stretchBltCoord(
    MGLDC *dst,
    MGLDC *src,
    int left,
    int top,
    int right,
    int bottom,
    int dstLeft,
    int dstTop,
    int dstRight,
    int dstBottom,
    int op)
```

Prototype In

mgraph.h

Parameters

<i>dst</i>	Destination device context
<i>src</i>	Source device context
<i>left</i>	Left coordinate of source image
<i>top</i>	Top coordinate of source image
<i>right</i>	Right coordinate of source image
<i>bottom</i>	Bottom coordinate of source image
<i>dstLeft</i>	Left coordinate of destination image
<i>dstTop</i>	Top coordinate of destination image
<i>dstRight</i>	Right coordinate of destination image
<i>dstBottom</i>	Bottom coordinate of destination image
<i>op</i>	Write mode to use during Blt

Description

Copies a block of bitmap data from one device context to another, stretching or shrinking the image as necessary to fit the destination rectangle for the destination device context.

The source and destination device context may be the same, however the source and destination rectangles may not overlap. This routine has been highly optimized for absolute maximum performance, so it will provide the fastest method of stretching bitmap data between device contexts, and can also be used to stretch bitmap data from a memory device context to a windowed device context.

This function will correctly handle StretchBlt's across device contexts with differing pixel depths, and will perform the necessary pixel format translation to convert from the source device to the destination device. Note that although the code to implement this is

highly optimized, this can be a time consuming operation so you should attempt to pre-convert all bitmaps to the current display device pixel format for maximum performance if possible.

When this function is called for 4 and 8 bit source bitmaps being copied to either 4 or 8bpp destination device contexts, MGL first checks if the color palettes for the source and destination bitmaps are the same. If they are not, MGL translates the pixel values from the source bitmap to the destination color palette, looking for the closest match color if an exact match is not found. In order to obtain maximum performance for blt'ing bitmaps in color index modes, you should ensure that the color palette in the source device matches the color palette in the destination device to avoid on the fly palette translation. If you know in advance that the palette is identical for a series of blit operations, you can turn off all identity palette checking in MGL with the *MGL_checkIdentityPalette* function.

When this function is called for 4 and 8 bit source bitmaps being copied to RGB destination device contexts, MGL will convert the pixels in the source bitmap using the source bitmap palette to map them to the destination pixel format. If however you know in advance that the palette for all source bitmaps is identical for a series of blit operations, you can use the *MGL_checkIdentityPalette* function to disable source palette translation. In this case the MGL will translate all color index bitmaps using the pre-defined color translation palette stored in the destination device context. You would then set the destination device context palette to the common palette for all blit operations using *MGL_setPalette*. If you are translating a lot of color index bitmaps, this will increase performance by avoiding the need to convert the palette entries to the destination pixel format for every blit operation.

The source and destination rectangles are clipped according to the current clipping rectangles for the source and destination device contexts respectively, however the zoom factor is determined using the unclipped source and destination rectangles.

See Also

MGL_bitBlit, *MGL_bitBlitCoord*, *MGL_srcTransBlit*, *MGL_srcTransBlitCoord*,
MGL_dstTransBlit, *MGL_dstTransBlitCoord*, *MGL_bitBlitPatt*, *MGL_bitBlitPattCoord*,
MGL_bitBlitFx, *MGL_bitBlitFxCoord*, *MGL_stretchBlit*, *MGL_stretchBlitCoord*,
MGL_stretchBlitFx, *MGL_stretchBlitFxCoord*, *MGL_copyPage*, *MGL_copyPageCoord*

MGL_stretchBltFx

Stretches a block of image data from one device context to another, while applying different effects in the process.

Declaration

```
void MGL_stretchBltFx(
    MGLDC dst,
    MGLDC src,
    rect_t srcRect,
    rect_t dstRect,
    bltfx_t *fx)
```

Prototype In

mgraph.h

Parameters

<i>dst</i>	Destination device context
<i>src</i>	Source device context
<i>srcRect</i>	Rectangle defining source image
<i>dstRect</i>	Rectangle defining destination image
<i>fx</i>	Information describing the effects to apply to the blit (<i>bltfx_t</i>)

Description

This function is the same as *MGL_stretchBltFxCoord*, however it takes entire rectangles as arguments instead of coordinates.

See Also

MGL_bitBlt, *MGL_bitBltCoord*, *MGL_srcTransBlt*, *MGL_srcTransBltCoord*,
MGL_dstTransBlt, *MGL_dstTransBltCoord*, *MGL_bitBltPatt*, *MGL_bitBltPattCoord*,
MGL_bitBltFx, *MGL_bitBltFxCoord*, *MGL_stretchBlt*, *MGL_stretchBltCoord*,
MGL_stretchBltFx, *MGL_stretchBltFxCoord*, *MGL_copyPage*, *MGL_copyPageCoord*

MGL_stretchBltFxCoord

Stretches a block of image data from one device context to another, while applying different effects in the process.

Declaration

```
void MGLAPI MGL_stretchBltFxCoord(
    MGLDC *dst,
    MGLDC *src,
    int left,
    int top,
    int right,
    int bottom,
    int dstLeft,
    int dstTop,
    int dstRight,
    int dstBottom,
    bltfx_t *fx)
```

Prototype In

mgraph.h

Parameters

<i>dst</i>	Destination device context
<i>src</i>	Source device context
<i>left</i>	Left coordinate of source image
<i>top</i>	Top coordinate of source image
<i>right</i>	Right coordinate of source image
<i>bottom</i>	Bottom coordinate of source image
<i>dstLeft</i>	Left coordinate of destination image
<i>dstTop</i>	Top coordinate of destination image
<i>dstRight</i>	Right coordinate of destination image
<i>dstBottom</i>	Bottom coordinate of destination image
<i>fx</i>	Information describing the effects to apply to the blit (<i>bltfx_t</i>)

Description

Copies a block of bitmap data from one device context to another, stretching or shrinking the image as necessary to fit the destination rectangle for the destination device context, while applying optional effects. This function behaves identically to the *MGL_bitBltFxCoord* function, except it includes the ability to stretch the bitmap at the same time. You can use the flags member of the *bltfx_t* structure to define whether stretching is done using nearest color stretching or if pixel interpolation will be used. If you don't specify any flags, nearest color stretching will be used.

This function will correctly handle effects Blt's across device contexts with differing pixel depths, and will perform the necessary pixel format translation to convert from the source device to the destination device. Note that although the code to implement this is highly optimized, this can be a time consuming operation so you should attempt to pre-

convert all bitmaps to the current display device pixel format for maximum performance if using this routine for sprite animation.

When this function is called for 4 and 8 bit source bitmaps being copied to either 4 or 8bpp destination device contexts, MGL first checks if the color palettes for the source and destination bitmaps are the same. If they are not, MGL translates the pixel values from the source bitmap to the destination color palette, looking for the closest match color if an exact match is not found. In order to obtain maximum performance for blt'ing bitmaps in color index modes, you should ensure that the color palette in the source device matches the color palette in the destination device to avoid on the fly palette translation. If you know in advance that the palette is identical for a series of blit operations, you can turn off all identity palette checking in MGL with the *MGL_checkIdentityPalette* function.

When this function is called for 4 and 8 bit source bitmaps being copied to RGB destination device contexts, MGL will convert the pixels in the source bitmap using the source bitmap palette to map them to the destination pixel format. If however you know in advance that the palette for all source bitmaps is identical for a series of blit operations, you can use the *MGL_checkIdentityPalette* function to disable source palette translation. In this case the MGL will translate all color index bitmaps using the pre-defined color translation palette stored in the destination device context. You would then set the destination device context palette to the common palette for all blit operations using *MGL_setPalette*. If you are translating a lot of color index bitmaps, this will increase performance by avoiding the need to convert the palette entries to the destination pixel format for every blit operation.

The source and destination rectangles are clipped according to the current clipping rectangles for the source and destination device contexts respectively.

Note: *This function is not designed to support overlapping source and destination rectangles on the same device context so if the source and destination rectangles overlap on the same device context, the results are undefined.*

See Also

MGL_bitBlit, MGL_bitBlitCoord, MGL_srcTransBlit, MGL_srcTransBlitCoord, MGL_dstTransBlit, MGL_dstTransBlitCoord, MGL_bitBlitPatt, MGL_bitBlitPattCoord, MGL_bitBlitFx, MGL_bitBlitFxCoord, MGL_stretchBlit, MGL_stretchBlitCoord, MGL_stretchBlitFx, MGL_stretchBlitFxCoord, MGL_copyPage, MGL_copyPageCoord

MGL_stretchBuffer

Stretches an offscreen buffer to the specified rectangle.

Declaration

```
void MGLAPI MGL_stretchBuffer(
    MGLDC *dc,
    int dstLeft,
    int dstTop,
    int dstRight,
    int dstBottom,
    MGLBUF *buf,
    int op)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to copy bufer to
<i>dstLeft</i>	Left coordinate to stretch buffer to
<i>dstTop</i>	Top coordinate to stretch buffer to
<i>dstRight</i>	Right coordinate to stretch buffer to
<i>dstBottom</i>	Bottom coordinate to stretch buffer to
<i>buf</i>	Buffer to display
<i>op</i>	Write mode to use when drawing buffer

Description

Stretches an offscreen buffer to the destination rectangle on the specififed device context.

Note: *This function will fail if you attempt to copy a buffer to a device context that is not the device context that the original buffer was allocated for.*

See Also

MGL_copyToBuffer, MGL_copyBitmapToBuffer, MGL_updateBufferCache, MGL_updateFromBufferCache, MGL_putBuffer, MGL_putBufferSection, MGL_putBufferSrcTrans, MGL_putBufferSrcTransSection, MGL_putBufferDstTrans, MGL_putBufferDstTransSection, MGL_putBufferPatt, MGL_putBufferPattSection, MGL_putBufferFx, MGL_putBufferFxSection, MGL_stretchBuffer, MGL_stretchBufferSection, MGL_stretchBufferFx, MGL_stretchBufferFxSection

MGL_stretchBufferFx

Stretches an offscreen buffer to the specified rectangle, while applying different effects in the process.

Declaration

```
void MGLAPI MGL_stretchBufferFx(
    MGLDC *dc,
    int dstLeft,
    int dstTop,
    int dstRight,
    int dstBottom,
    MGLBUF *buf,
    bltfx_t *fx)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to copy bufer to
<i>dstLeft</i>	Left coordinate to stretch buffer to
<i>dstTop</i>	Top coordinate to stretch buffer to
<i>dstRight</i>	Right coordinate to stretch buffer to
<i>dstBottom</i>	Bottom coordinate to stretch buffer to
<i>buf</i>	Buffer to display
<i>fx</i>	Information describing the effects to apply to the blit (<i>bltfx_t</i>)

Description

Stretches an offscreen buffer to the destination rectangle on the specififed device context, while applying optional effects. You can use the flags member of the *bltfx_t* structure to define whether stretching is done using nearest color stretching or if pixel interpolation will be used. If you don't specify any flags, nearest color stretching will be used.

Note: *This function will fail if you attempt to copy a buffer to a device context that is not the device context that the original buffer was allocated for.*

See Also

MGL_copyToBuffer, MGL_copyBitmapToBuffer, MGL_updateBufferCache, MGL_updateFromBufferCache, MGL_putBuffer, MGL_putBufferSection, MGL_putBufferSrcTrans, MGL_putBufferSrcTransSection, MGL_putBufferDstTrans, MGL_putBufferDstTransSection, MGL_putBufferPatt, MGL_putBufferPattSection, MGL_putBufferFx, MGL_putBufferFxSection, MGL_stretchBuffer, MGL_stretchBufferSection, MGL_stretchBufferFx, MGL_stretchBufferFxSection

MGL_stretchBufferFxSection

Stretches a section of an offscreen buffer to the specified device context, while applying different effects in the process.

Declaration

```
void MGLAPI MGL_stretchBufferFxSection(
    MGLDC *dc,
    int left,
    int top,
    int right,
    int bottom,
    int dstLeft,
    int dstTop,
    int dstRight,
    int dstBottom,
    MGLBUF *buf,
    bltfx_t *fx)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to copy buffer to
<i>left</i>	Left coordinate of section to stretch
<i>top</i>	Top coordinate of section to stretch
<i>right</i>	Right coordinate of section to stretch
<i>bottom</i>	Bottom coordinate of section to stretch
<i>dstLeft</i>	Left coordinate to stretch buffer to
<i>dstTop</i>	Top coordinate to stretch buffer to
<i>dstRight</i>	Right coordinate to stretch buffer to
<i>dstBottom</i>	Bottom coordinate to stretch buffer to
<i>buf</i>	Buffer to display
<i>fx</i>	Information describing the effects to apply to the blit (<i>bltfx_t</i>)

Description

Stretches a section of an offscreen buffer to the destination rectangle on the specified device context, while applying optional effects. You can use the flags member of the *bltfx_t* structure to define whether stretching is done using nearest color stretching or if pixel interpolation will be used. If you don't specify any flags, nearest color stretching will be used.

Note: *This function will fail if you attempt to copy a buffer to a device context that is not the device context that the original buffer was allocated for.*

See Also

MGL_copyToBuffer, MGL_copyBitmapToBuffer, MGL_updateBufferCache, MGL_updateFromBufferCache, MGL_putBuffer, MGL_putBufferSection,

*MGL_putBufferSrcTrans, MGL_putBufferSrcTransSection, MGL_putBufferDstTrans,
MGL_putBufferDstTransSection, MGL_putBufferPatt, MGL_putBufferPattSection,
MGL_putBufferFx, MGL_putBufferFxSection, MGL_stretchBuffer, MGL_stretchBufferSection,
MGL_stretchBufferFx, MGL_stretchBufferFxSection*

MGL_stretchBufferSection

Stretches a section of an offscreen buffer to the specified device context.

Declaration

```
void MGLAPI MGL_stretchBufferSection(
    MGLDC *dc,
    int left,
    int top,
    int right,
    int bottom,
    int dstLeft,
    int dstTop,
    int dstRight,
    int dstBottom,
    MGLBUF *buf,
    int op)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Device context to copy buffer to
<i>left</i>	Left coordinate of section to stretch
<i>top</i>	Top coordinate of section to stretch
<i>right</i>	Right coordinate of section to stretch
<i>bottom</i>	Bottom coordinate of section to stretch
<i>dstLeft</i>	Left coordinate to stretch buffer to
<i>dstTop</i>	Top coordinate to stretch buffer to
<i>dstRight</i>	Right coordinate to stretch buffer to
<i>dstBottom</i>	Bottom coordinate to stretch buffer to
<i>buf</i>	Buffer to display
<i>op</i>	Write mode to use when drawing buffer

Description

Stretches a section of an offscreen buffer to the destination rectangle on the specified device context.

Note: *This function will fail if you attempt to copy a buffer to a device context that is not the device context that the original buffer was allocated for.*

See Also

MGL_copyToBuffer, MGL_copyBitmapToBuffer, MGL_updateBufferCache, MGL_updateFromBufferCache, MGL_putBuffer, MGL_putBufferSection, MGL_putBufferSrcTrans, MGL_putBufferSrcTransSection, MGL_putBufferDstTrans, MGL_putBufferDstTransSection, MGL_putBufferPatt, MGL_putBufferPattSection, MGL_putBufferFx, MGL_putBufferFxSection, MGL_stretchBuffer, MGL_stretchBufferSection, MGL_stretchBufferFx, MGL_stretchBufferFxSection

MGL_surfaceAccessType

Return the direct surface access flags.

Declaration

```
int MGLAPI MGL_surfaceAccessType(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Device context of interest

Return Value

Direct surface access flags for the device context.

Description

This function returns the direct surface access flags for the specified device context, which can be used to determine if the surface for the device context is directly accessible, and if the surface has been virtualized in software. The access flags returned are enumerated in *MGL_surfaceAccessType*.

If the surface access flags is MGL_VIRTUAL_ACCESS, this means that the surface for the device can be directly accessed, however the surface is actually virtualized in software using a page fault handler for SuperVGA devices that do not have a real hardware linear framebuffer. If the surface is virtualized, you must ensure that when you directly access the surface you do so on BYTE, WORD and DWORD aligned boundaries. If you access it on a non-aligned boundary across a page fault, you will cause an infinite page fault loop to occur. If the surface access flags is MGL_NO_ACCESS the framebuffer will be banked and if you wish to rasterize directly to it you will need to use the SVGA_setBank functions to change banks. In banked modes the surface pointer points to the start of the banked framebuffer window (ie: 0xA0000).

MGL_suspend

Suspend low level interrupt handling.

Declaration

```
void MGLAPI MGL_suspend(void)
```

Prototype In

mgldos.h

Description

This function suspends the low level interrupt handling code used by the SciTech MGL when it is initialized since MGL takes over the keyboard and mouse interrupt handlers to manage it's own event queue. If you wish to shell out to DOS or to spawn another application program temporarily, you must call this function to suspend interrupt handling or else the spawned application will not be able to access the keyboard and mouse correctly.

See Also

MGL_resume

MGL_swapBuffers

Swaps the currently active front and back buffers for a display device context.

Declaration

```
void MGLAPI MGL_swapBuffers(
    MGLDC *dc,
    int waitVRT)
```

Prototype In

mgraph.h

Parameters

<i>dc</i>	Display device context
<i>waitVRT</i>	Wait for retrace flag (<i>MGL_waitVRTFlagType</i>)

Description

This function swaps the currently active front and back buffers. This routine should only be called after the *MGL_doubleBuffer* has been called to initialize the double buffering for the device context. Once double buffering has been set up, all output from MGL will go to the current offscreen buffer, and the output can be made visible by calling this routine. This routine is the standard technique used to achieve smooth animation.

When the visible display buffer is changed, you should normally allow MGL to sync to the vertical retrace by passing a value of *MGL_waitVRT* in the *waitVRT* parameter to ensure that the change occurs in the correct place, and that you don't get flicker effects on the display.

You may however want to turn off the vertical retrace synching if you are synching up with the retrace period using some other means by passing a value of *MGL_dontWait* to the *waitVRT* parameter. This is also useful if you are measuring the performance of your application and you want it to run at full speed without the overhead of waiting for the vertical retrace.

Note: *This function only implements double buffering so do not pass a value of MGL_tripleBuffer in the waitVRT parameter. If you need triple buffering support, use the MGL_setActivePage and MGL_setVisual page functions directly to implement this.*

See Also

MGL_doubleBuffer, MGL_singleBuffer

MGL_textBounds

Compute the bounding box for a text string.

Declaration

```
void MGLAPI MGL_textBounds (
    int x,
    int y,
    const char *str,
    rect_t *bounds)
```

Prototype In

mgraph.h

Parameters

<i>x</i>	x coordinate string would be drawn at
<i>y</i>	y coordinate string would be drawn at
<i>str</i>	String to measure
<i>bounds</i>	Place to store the computed bounds

Description

This function computes the bounding box that fits tightly around a text string drawn at a specified location on the current device context. This routine correctly computes the bounding rectangle for the string given the current text justification, size and direction settings.

See Also

MGL_textBounds_W, *MGL_textHeight*, *MGL_textWidth*, *MGL_textWidth_W*

MGL_textBounds_W

Compute the bounding box for a wide character string.

Declaration

```
void MGLAPI MGL_textBounds_W(
    int x,
    int y,
    const wchar_t *str,
    rect_t *bounds)
```

Prototype In

mgraph.h

Parameters

<i>x</i>	x coordinate string would be drawn at
<i>y</i>	y coordinate string would be drawn at
<i>str</i>	Wide character string to measure
<i>bounds</i>	Place to store the computed bounds

Description

This function computes the bounding box that fits tightly around a text string drawn at a specified location on the current device context. This routine correctly computes the bounding rectangle for the string given the current text justification, size and direction settings.

This function is the same as *MGL_textBounds*, but provides support for Unicode wide characters (for far-east languages).

Note: *Wide character fonts are only supported for bitmap and TrueType fonts. Vector fonts are not supported via this function.*

See Also

MGL_textBounds, *MGL_textHeight*, *MGL_textWidth*, *MGL_textWidth_W*

MGL_textHeight

Returns the height of the current font in pixels.

Declaration

```
int MGLAPI MGL_textHeight(void)
```

Prototype In

mgraph.h

Return Value

Height of the current font in pixels

Description

Returns the height of the currently active font in pixels. This includes any scaling transformations that are applied to the font and will be as accurate as possible at the resolution of the display device.

See Also

MGL_textWidth, MGL_drawStr, MGL_getCharMetrics, MGL_getFontMetrics

MGL_textWidth

Returns the width of the character string in pixels.

Declaration

```
int MGLAPI MGL_textWidth(  
    const char *str)
```

Prototype In

mgraph.h

Parameters

str Character string to measure

Return Value

Width of the character string in pixels

Description

Returns the width of the specified character string using the dimensions of the currently active font in pixels. This includes any scaling transformations that are applied to the font and will be as accurate as possible at the resolution of the display device.

See Also

MGL_textWidth_W, *MGL_textHeight*, *MGL_drawStr*, *MGL_drawStr_W*,
MGL_getCharMetrics, *MGL_getCharMetrics_W*, *MGL_getFontMetrics*

MGL_textWidth_W

Returns the width of the wide character string in pixels.

Declaration

```
int MGLAPI MGL_textWidth_W(  
    const wchar_t *str)
```

Prototype In

mgraph.h

Parameters

str Wide character string to measure

Return Value

Width of the wide character string in pixels

Description

Returns the width of the specified character string using the dimensions of the currently active font in pixels. This includes any scaling transformations that are applied to the font and will be as accurate as possible at the resolution of the display device.

This function is the same as *MGL_textWidth*, but provides support for Unicode wide characters (for far-east languages).

Note: *Wide character fonts are only supported for bitmap and TrueType fonts. Vector fonts are not supported via this function.*

See Also

MGL_textWidth, *MGL_textHeight*, *MGL_drawStr*, *MGL_drawStr_W*, *MGL_getCharMetrics*, *MGL_getCharMetrics_W*, *MGL_getFontMetrics*

MGL_traverseRegion

Traverses a region for all rectangles in definition.

Declaration

```
void MGLAPI MGL_traverseRegion(
    region_t *rgn,
    rgncallback_t doRect)
typedef void (MGLAPIP rgncallback_t)(const rect_t *r)
```

Prototype In

mgraph.h

Parameters

<i>rgn</i>	Region to traverse
<i>doRect</i>	Callback function to call for every rectangle processed

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

MGL_diffRegion, MGL_unionRegion, MGL_sectRegion

MGL_underScoreLocation

Returns the location to begin drawing an underscore for the font.

Declaration

```
void MGLAPI MGL_underScoreLocation(  
    int *x,  
    int *y,  
    const char *str)
```

Prototype In

mgraph.h

Parameters

<i>x</i>	x coordinate to be passed to <i>MGL_drawStrXY</i>
<i>y</i>	y coordinate to be passed to <i>MGL_drawStrXY</i>
<i>str</i>	String to measure

Description

This function takes an (x,y) location that would normally be used to draw a string with *MGL_drawStrXY*, and adjusts the coordinates to begin at the under score location for the current font, in the current drawing attributes. Thus the entire character string can be underlined by drawing a line starting at the computed underscore location and extending for *MGL_textWidth* pixels in length.

See Also

MGL_underScoreLocation_W, *MGL_drawStrXY*, *MGL_textWidth*

MGL_underScoreLocation_W

Returns the location to begin drawing an underscore for the font.

Declaration

```
void MGLAPI MGL_underScoreLocation_W(
    int *x,
    int *y,
    const wchar_t *str)
```

Prototype In

mgraph.h

Parameters

<i>x</i>	x coordinate to be passed to <i>MGL_drawStrXY</i>
<i>y</i>	y coordinate to be passed to <i>MGL_drawStrXY</i>
<i>str</i>	Wide character string to measure

Description

This function takes an (x,y) location that would normally be used to draw a string with *MGL_drawStrXY*, and adjusts the coordinates to begin at the under score location for the current font, in the current drawing attributes. Thus the entire character string can be underlined by drawing a line starting at the computed underscore location and extending for *MGL_textWidth* pixels in length.

This function is the same as *MGL_underScoreLocation*, but provides support for Unicode wide characters (for far-east languages).

Note: *Wide character fonts are only supported for bitmap and TrueType fonts. Vector fonts are not supported via this function.*

See Also

MGL_underScoreLocation, *MGL_drawStrXY*, *MGL_textWidth*

MGL_unionRect

Computes the union of two rectangles.

Declaration

```
void MGLAPI MGL_unionRect(  
    rect_t r1,  
    rect_t r2,  
    rect_t *d)
```

Prototype In

mgraph.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

MGL_unionRectCoord, *MGL_sectRect*

MGL_unionRectCoord

Computes the union of two rectangles.

Declaration

```
void MGLAPI MGL_unionRectCoord(
    int left1,
    int top1,
    int right1,
    int bottom1,
    int left2,
    int top2,
    int right2,
    int bottom2,
    rect_t *d)
```

Prototype In

mgraph.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

MGL_sectRect

MGL_unionRegion

Computes the Boolean union of two regions.

Declaration

```
ibool MGLAPI MGL_unionRegion(
    region_t *r1,
    const region_t *r2)
```

Prototype In

mgraph.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

MGL_diffRegion, MGL_sectRegion, MGL_unionRegionOfs, MGL_unionRegionRect

MGL_unionRegionOfs

Computes the Boolean union of two regions and offsets the result.

Declaration

```
ibool MGLAPI MGL_unionRegionOfs(
    region_t *r1,
    const region_t *r2,
    int xOffset,
    int yOffset)
```

Prototype In

mgraph.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>xOffset</i>	Offset to add to all x coordinates in region 2
<i>yOffset</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

MGL_diffRegion, MGL_sectRegion, MGL_unionRegion, MGL_unionRegionRect

MGL_unionRegionRect

Computes the Boolean union of a region and a rectangle.

Declaration

```
ibool MGLAPI MGL_unionRegionRect(
    region_t *r1,
    const rect_t *r2)
```

Prototype In

mgraph.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 *MGL_unionRegion* if the region to be unioned is a simple rectangle rather than a complex region.

See Also

MGL_diffRegion, *MGL_sectRegion*, *MGL_unionRegion*, *MGL_unionRegionOfs*

MGL_unloadBitmap

Unloads a bitmap file from memory.

Declaration

```
void MGLAPI MGL_unloadBitmap(  
    bitmap_t *bitmap)
```

Prototype In

mgraph.h

Parameters

bitmap Pointer to bitmap to unload

Description

Unloads the specified bitmap file from memory, and frees up all the system resources associated with this bitmap.

See Also

MGL_loadBitmap

MGL_unloadCursor

Unloads a cursor file from memory.

Declaration

```
void MGLAPI MGL_unloadCursor(  
    cursor_t *cursor)
```

Prototype In

mgraph.h

Parameters

cursor Pointer to cursor to unload

Description

Unloads the specified cursor file from memory, and frees up all the system resources associated with this cursor.

See Also

MGL_loadCursor

MGL_unloadFont

Unloads a font file from memory.

Declaration

```
void MGLAPI MGL_unloadFont(  
    font_t *font)
```

Prototype In

mgraph.h

Parameters

font Pointer to font to unload

Description

Unloads the specified font file from memory, and frees up all the system resources associated with this font.

Note: *This function is now obsolete. Please use the font library functions.*

See Also

MGL_loadFont, MGL_openFontLib, MGL_unloadFontLib, MGL_loadFontInstance, MGL_unloadFontInstance

MGL_unloadFontInstance

Unloads a font instance from memory.

Declaration

```
void MGLAPI MGL_unloadFontInstance(  
    font_t *font)
```

Prototype In

mgraph.h

Parameters

font Pointer to font instance to unload

Description

Unloads the specified font instance from memory, and frees up all the system resources associated with this font instance.

See Also

MGL_openFontLib, *MGL_unloadFontLib*, *MGL_loadFontInstance*

MGL_unloadIcon

Unloads an icon file from memory.

Declaration

```
void MGLAPI MGL_unloadIcon(  
    icon_t *icon)
```

Prototype In

mgraph.h

Parameters

icon Pointer to icon to unload

Description

Unloads the specified icon file from memory, and frees up all the system resources associated with this icon.

See Also

MGL_loadIcon

MGL_unlockBuffer

Unlock a buffer after direct surface access

Declaration

```
void MGLAPI MGL_unlockBuffer(  
    MGLBUF *buf)
```

Prototype In

mgraph.h

Parameters

buf MGL buffer to unlock

Description

This function unlocks a buffer after the application has completed direct surface access on the buffer.

See Also

MGL_lockBuffer

MGL_unpackColor

Unpacks a packed MGL color value into RGB components.

Declaration

```
void MGLAPI MGL_unpackColor(
    pixel_format_t *pf,
    color_t color,
    uchar *R,
    uchar *G,
    uchar *B)
```

Prototype In

mgraph.h

Parameters

<i>pf</i>	Pixel format to use for unpacking
<i>color</i>	Color to unpack
<i>R</i>	Place to store extracted red component
<i>G</i>	Place to store extracted green component
<i>B</i>	Place to store extracted blue component

Description

This function takes a packed color value in the correct format for the specified pixel format and extracts the red, green and blue components. Note that the color values may not be the same as when you packed them with *MGL_packColor* if the pixel format is a 15 or 16 bit format because of loss of precision. The values are scaled back into the normal 24 bit RGB space.

See Also

MGL_unpackColorFast, *MGL_unpackColorExt*, *MGL_packColor*, *MGL_getPixelFormat*

MGL_unpackColorExt

Unpacks a packed MGL color value into RGB components.

Declaration

```
void MGLAPI MGL_unpackColorExt(
    pixel_format_t *pf,
    color_t color,
    uchar *A,
    uchar *R,
    uchar *G,
    uchar *B)
```

Prototype In

mgraph.h

Parameters

<i>pf</i>	Pixel format to use for unpacking
<i>color</i>	Color to unpack
<i>A</i>	Place to store extracted alpha component
<i>R</i>	Place to store extracted red component
<i>G</i>	Place to store extracted green component
<i>B</i>	Place to store extracted blue component

Description

This function takes a packed color value in the correct format for the specified pixel format and extracts the red, green and blue components. Note that the color values may not be the same as when you packed them with *MGL_packColor* if the pixel format is a 15 or 16 bit format because of loss of precision. The values are scaled back into the normal 24 bit RGB space.

See Also

MGL_unpackColorFast, *MGL_unpackColorExt*, *MGL_packColor*, *MGL_getPixelFormat*

MGL_unpackColorFast

Unpacks a packed MGL color value into RGB components.

Declaration

```
void MGL_unpackColorFast(  
    pixel_format_t *pf,  
    color_t color,  
    uchar R,  
    uchar G,  
    uchar B)
```

Prototype In

mgraph.h

Parameters

<i>pf</i>	Pixel format to use for unpacking
<i>color</i>	Color to unpack
<i>R</i>	Place to store extracted red component
<i>G</i>	Place to store extracted green component
<i>B</i>	Place to store extracted blue component

Description

This function is the same as *MGL_unpackColor*, however it is implemented as a macro and hence is more efficient.

See Also

MGL_unpackColorExt, *MGL_unpackColor*, *MGL_packColor*, *MGL_getPixelFormat*

MGL_unpackColorFastExt

Unpacks a packed MGL color value into RGB components.

Declaration

```
void MGL_unpackColorFastExt(  
    pixel_format_t *pf,  
    color_t color,  
    uchar A,  
    uchar R,  
    uchar G,  
    uchar B)
```

Prototype In

mgraph.h

Parameters

<i>pf</i>	Pixel format to use for unpacking
<i>color</i>	Color to unpack
<i>A</i>	Place to store extracted alpha component
<i>R</i>	Place to store extracted red component
<i>G</i>	Place to store extracted green component
<i>B</i>	Place to store extracted blue component

Description

This function is the same as *MGL_unpackColor*, however it is implemented as a macro and hence is more efficient.

See Also

MGL_unpackColorExt, *MGL_unpackColor*, *MGL_packColor*, *MGL_getPixelFormat*

MGL_updateBufferCache

Updates the system cache from the video memory buffer contents

Declaration

```
void MGLAPI MGL_updateBufferCache(  
    MGLBUF *buf)
```

Prototype In

mgraph.h

Parameters

buf MGL 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

MGL_updateFromBufferCache, *MGL_putBuffer*

MGL_updateFromBufferCache

Updates the video memory buffer contents from system memory cache

Declaration

```
void MGLAPI MGL_updateFromBufferCache(  
    MGLBUF *buf)
```

Prototype In

mgraph.h

Parameters

buf MGL 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 *MGL_updateBufferCache*.

See Also

MGL_updateBufferCache, *MGL_putBuffer*

MGL_useFont

Sets the currently active font.

Declaration

```
ibool MGLAPI MGL_useFont (  
    font_t *font)
```

Prototype In

mgraph.h

Parameters

font New font to use

Return Value

True if the font was valid and selected, false if not.

Description

Selects the specified font as the currently active font for the active device context. If the font data is invalid, the MGL result flag is set and the routine will return false.

Do not unload a font file if it is currently in use by MGL!

See Also

MGL_drawStr, MGL_loadFont, MGL_unloadFont

MGL_usePenBitmapPattern

Sets the currently active bitmap pattern.

Declaration

```
void MGLAPI MGL_usePenBitmapPattern(  
    int index)
```

Prototype In

mgraph.h

Parameters

index Index of the bitmap pattern to make active

Description

This function sets the currently active bitmap pattern used when rendering patterned primitives in the MGL_BITMAP_TRANSPARENT and MGL_BITMAP_OPQAUE pen styles. The pattern must already have been downloaded with a call to the *MGL_setPenBitmapPattern* function.

See Also

MGL_setPenBitmapPattern, *MGL_getPenBitmapPattern*, *MGL_setPenPixmapPattern*, *MGL_setPenStyle*

MGL_usePenPixmapPattern

Sets the currently active pixmap pattern.

Declaration

```
void MGLAPI MGL_usePenPixmapPattern(  
    int index)
```

Prototype In

mgraph.h

Parameters

index Index of the pixmap pattern to make active

Description

This function sets the currently active bitmap pattern used when rendering patterned primitives in the MGL_PIXMAP pen style. The pattern must already have been downloaded with a call to the *MGL_setPenPixmapPattern* function.

See Also

MGL_setPenPixmapPattern, *MGL_getPenPixmapPattern*, *MGL_setPenBitmapPattern*,
MGL_setPenStyle

MGL_vSync

Waits for the vertical sync for a display device context.

Declaration

```
ibool MGLAPI MGL_vSync(  
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc Display device context of interest

Return Value

True on success, false if function not available on hardware device.

Description

This function will wait until the next vertical sync comes along for the current fullscreen graphics mode, before returning.

Note: *This function may not be available on all display devices, so make sure you check the return value to see if the function is implemented and available.*

See Also

MGL_isVSync, MGL_getCurrentScanLine

MGL_vecFontEngine

Generates the commands to draw a vectored font.

Declaration

```
ibool MGLAPI MGL_vecFontEngine(
    int x,
    int y,
    const char *str,
    void (MGLAPIP move)(int x,int y),
    void (MGLAPIP draw)(int x,int y))
```

Prototype In

mgraph.h

Parameters

<i>x</i>	x coordinate to start drawing text at
<i>y</i>	y coordinate to start drawing text at
<i>str</i>	Character string to draw
<i>move</i>	Routine to call to perform a move operation
<i>draw</i>	Routine to call to perform a draw operation

Return Value

True if the string is correctly rasterized, false if font is not a vector font.

Description

This function calls a set of user supplied routines to rasterize the characters in a vector font. This allows the vector fonts to be drawn in 2D or 3D floating point coordinate systems by transforming each of the coordinates required to draw each character by any arbitrary transformation, or in any coordinate system that the users desires.

The move routine is called to move the cursor to a new location, and the draw routine is used to perform a draw operation from the current location to the specified location. Each character in the vector font is started with a move operation.

Note that the coordinates passed to the move and draw routines will be offset from the point (x,y), where the point (x,y) is the origin of the first character (i.e. it lies on its baseline). Note also that the coordinates will be relative to the origin with the origin at the lower left corner of each character (i.e. inverse of normal device coordinate y-axis values).

This routine does not honor the standard scaling factors, but simply draws the characters with a size of (1,1,1,1) (because scaling will be done by the user supplied move and draw routines).

If the passed font is not a valid vector font, this routine returns false.

See Also

MGL_drawStr, MGL_useFont

MGL_wmBeginPaint

Creates DC suitable for painting on window

Declaration

```
MGLDC* MGLAPI MGL_wmBeginPaint(
    window_t *wnd)
```

Prototype In

mgraph.h

Parameters

wnd the window to paint on

Return Value

Returns device context that you can paint on.

Description

If you can't use painter function to paint on a window for some reason, MGL provides this function. It returns a DC prepared for drawing on it (specifically, clipping region is set to clip off anything that is not in window's visible part and coordinate system is changed to be local to the window).

You must call *MGL_wmEndPaint* when you're done with painting on the window.

Note: *Avoid setting clipping region on returned DC. If you must do so, make sure you intersect your clipping region with current clipping region of the DC!*

Note: *This function automatically hides mouse pointer if necessary to avoid occurrence rendering artifacts.*

Note: *The effect of MGL_wmBeginPaint/MGL_wmEndPaint drawing is temporary, MGL_wmUpdateDC may (and probably will) redraw it using the painter later.*

See Also

MGL_wmUpdateDC, MGL_wmSetWindowPainter, MGL_wmEndPaint

MGL_wmCaptureEvents

Captures events and redirects them to specific window.

Declaration

```
void MGLAPI MGL_wmCaptureEvents (
    window_t *wnd,
    ulong mask,
    int id)
```

Prototype In mgraph.h

Parameters

<i>wnd</i>	window
<i>mask</i>	mask of events to capture (for <i>event_t.what</i>)
<i>id</i>	arbitrary user-specified ID of the capture entry (used when removing the entry with <i>MGL_wmUncaptureEvents</i>)

Description

It is sometimes useful to redirect certain kinds of events to a particular window instead of to the one under mouse pointer (e.g. you may want to send all keyboard events to the window that has focus in a program with Windows feel). This function allows you to redirect all events whose mask matches *event_t.what* of the event (i.e. *event_t.what* & mask != 0) to the *wnd* window.

You may capture events several times, to the same window or to different ones. Capture entries behave like stack: the ones that were added later take precedence over earlier registered ones.

The ID is used to uniquely identify the entry when removing it and you must ensure they are unique among all captures that are active at the same time.

See Also

MGL_wmProcessEvent, *MGL_wmPushWindowEventHandler*,
MGL_wmPopWindowEventHandler, *MGL_wmRemoveWindowEventHandler*,
MGL_wmPushGlobalEventHandler, *MGL_wmPopGlobalEventHandler*,
MGL_wmRemoveGlobalEventHandler, *MGL_wmUncaptureEvents*

MGL_wmCoordGlobalToLocal

Converts between local and global coordinates.

Declaration

```
void MGLAPI MGL_wmCoordGlobalToLocal (
    window_t *wnd,
    int x,
    int y,
    int *xLocal,
    int *yLocal)
```

Prototype In

mgraph.h

Parameters

<i>wnd</i>	the window
<i>x</i>	X coordinate in global space
<i>y</i>	Y coordinate in global space
<i>xLocal</i>	variable to store X coordinate in local space
<i>yLocal</i>	variable to store X coordinate in local space

Description

This function converts point expressed in global coordinates (i.e. relative to device context's upper left corner) into coordinates relative to given window's upper left corner.

wnd may be a window arbitrary deep in windows hierarchy,
MGL_wmCoordGlobalToLocal descends the hierarchy recursively.

See Also

MGL_wmCoordGlobalToLocal, *MGL_wmSetWindowPosition*

MGL_wmCoordLocalToGlobal

Converts between global and local coordinates.

Declaration

```
void MGLAPI MGL_wmCoordLocalToGlobal (
    window_t *wnd,
    int x,
    int y,
    int *xGlobal,
    int *yGlobal)
```

Prototype In

mgraph.h

Parameters

<i>wnd</i>	the window
<i>x</i>	X coordinate in local space
<i>y</i>	Y coordinate in local space
<i>xGlobal</i>	variable to store X coordinate in global space
<i>yGlobal</i>	variable to store X coordinate in global space

Description

This function converts point expressed in coordinates relative to given window's upper left corner to global coordinates (i.e. relative to device context's upper left corner).

wnd may be a window arbitrary deep in windows hierarchy, *MGL_wmCoordLocalToGlobal* descends the hierarchy recursively.

See Also

MGL_wmCoordLocalToGlobal, *MGL_wmSetWindowPosition*

MGL_wmCreate

Creates window manager object and attaches it to device context.

Declaration

```
winmng_t* MGLAPI MGL_wmCreate(
    MGLDC *dc)
```

Prototype In

mgraph.h

Parameters

dc display device context to associate with window manager.

Return Value

Window manager object or NULL on error

Description

This function creates window manager object. MGL WM provides functionality similar to that of Xlib, i.e. bare minimum needed to implement windowing environment on top of SciTech MGL. That is, it manages hierarchy of rectangular windows, takes care of proper repainting (but you must provide painter functions for all windows) and clipping and distributes input events among the windows.

In addition to allocating *winmng_t* object, this function also creates the root window (which is top-level window that covers entire DC area and that is at the top of windows hierarchy). Root window is accessible via *MGL_wmGetRootWindow*. It also sets the root window visible and shows mouse cursor.

You must create *winmng_t* object with this function before you can use MGL WM.

Note: *You shouldn't draw to the device context associated with window manager yourself. Use MGL WM functions *MGL_wmBeginPaint* and *MGL_wmSetWindowPainter* instead.*

Note: *MGL_wmCreate doesn't set root window's painter method. You must do it yourself, preferably immediately after creating the *winmng_t* object.*

See Also

MGL_wmDestroy, *MGL_wmCreateWindow*, *MGL_wmUpdateDC*, *MGL_wmGetRootWindow*, *MGL_wmProcessEvent*

MGL_wmCreateWindow

Creates new window.

Declaration

```
window_t *MGLAPI MGL_wmCreateWindow(
    winmng_t *wm,
    window_t *parent,
    int x,
    int y,
    size_t width,
    size_t height)
```

Prototype In mgraph.h

Parameters

<i>wm</i>	window manager that will manage this window
<i>parent</i>	parent window
<i>x</i>	X coordinate of initial window position
<i>y</i>	Y coordinate of initial window position
<i>width</i>	initial window width
<i>height</i>	initial window height

Return Value

The created window.

Description

Creates new window and assigns it to given window manager. The window is created as invisible, you must call *MGL_wmShowWindow* to show it. You must also call *MGL_wmSetWindowPainter* to set painting callback, otherwise the window will be transparent.

If parent is NULL, the window is created as top level one and its parent is set to window manager's root window.

See Also

MGL_wmCreate, *MGL_wmDestroyWindow*, *MGL_wmSetWindowPosition*,
MGL_wmSetWindowPainter, *MGL_wmSetWindowCursor*, *MGL_wmSetWindowFlags*,
MGL_wmSetWindowUserData, *MGL_wmSetWindowDestructor*,
MGL_wmGetWindowParent, *MGL_wmCoordGlobalToLocal*, *MGL_wmCoordLocalToGlobal*,
MGL_wmGetWindowAtPosition, *MGL_wmReparentWindow*, *MGL_wmLowerWindow*,
MGL_wmRaiseWindow, *MGL_wmShowWindow*, *MGL_wmInvalidateWindow*,
MGL_wmBeginPaint, *MGL_wmPushWindowEventHandler*,
MGL_wmPopWindowEventHandler, *MGL_wmCaptureEvents*, *MGL_wmUncaptureEvents*

MGL_wmDestroy

Destroys window manager.

Declaration

```
void MGLAPI MGL_wmDestroy(  
    winmng_t *wm)
```

Prototype In

mgraph.h

Parameters

wm window manager to destroy

Description

This function destroys the window manager object and all windows managed by it. It doesn't destroy associated device context.

You must call this function before you destroy associated DC.

See Also

MGL_wmCreate

MGL_wmDestroyWindow

Destroys the window.

Declaration

```
void MGLAPI MGL_wmDestroyWindow(  
    window_t *wnd)
```

Prototype In

mgraph.h

Parameters

wnd the window to destroy

Description

Window destruction happens in two stages: first, the destructor is called if one was previously set with *MGL_wmSetWindowDestructor* and second, all children are destroyed recursively.

See Also

MGL_wmCreateWindow, *MGL_wmSetWindowDestructor*

MGL_wmEndPaint

Finishes painting on window

Declaration

```
void MGLAPI MGL_wmEndPaint(  
    window_t *wnd)
```

Prototype In

mgraph.h

Parameters

wnd the window you painted on

Description

This function must be called after using *MGL_wmBeginPaint* on the window. It restores mouse cursor and device context's attributes.

See Also

MGL_wmUpdateDC, *MGL_wmSetWindowPainter*, *MGL_wmBeginPaint*

MGL_wmGetRootWindow

Returns pointer to window manager's root window.

Declaration

```
window_t* MGLAPI MGL_wmGetRootWindow(
    winmng_t *wm)
```

Prototype In

mgraph.h

Parameters

wm the window manager

Return Value

The root window.

Description

This function returns pointer to window manager's root window. The root window is a special window several notable properties: You don't create or destroy it directly, *MGL_wmCreate* and *MGL_wmDestroy* do it. It covers entire device context and all windows are its children (or grand-children or grand-grand-children and so on). Windows created with NULL parent are direct children of the root window.

Note: *There is neither default event handling nor painter set to root window by MGL_wmCreate. You have to set at least the painter yourself.*

See Also

MGL_wmCreate, MGL_wmDestroy

MGL_wmGetWindowAtPosition

Finds window at given coordinates.

Declaration

```
window_t* MGLAPI MGL_wmGetWindowAtPosition(
    winmng_t *wm,
    int x,
    int y)
```

Prototype In

mgraph.h

Parameters

<i>wm</i>	the window manager
<i>x</i>	X coordinate (global)
<i>y</i>	Y coordinate (global)

Return Value

Pointer to window at given position.

Description

The function returns window that draws itself at position (x,y). It properly handles hidden and overlapping windows. The position is expressed in global coordinates, i.e. relative to device context.

MGL_wmGetWindowAtPosition can never return NULL; if no other window is found, pointer to wm's root window is returned;

See Also

MGL_wmCoordLocalToGlobal, *MGL_wmSetWindowPosition*

MGL_wmGetWindowFlags

Returns currently set window flags.

Declaration

```
long MGLAPI MGL_wmGetWindowFlags(  
    window_t *wnd)
```

Prototype In

mgraph.h

Parameters

wnd the window

Return Value

Window's flags.

See Also

MGL_wmSetWindowFlags

MGL_wmGetWindowParent

Returns parent window.

Declaration

```
window_t* MGLAPI MGL_wmGetWindowParent(  
    window_t *wnd)
```

Prototype In

mgraph.h

Parameters

wnd the window

Return Value

Pointer to parent window or NULL.

Description

The function returns pointer to parent window, i.e. the window whose this window is child.

If the window was created with NULL parent passed to *MGL_wmCreateWindow*, this function does not return NULL. Instead, it returns pointer to WM's root window.

MGL_wmGetWindowParent(MGL_wmGetRootWindow(wm)) returns NULL.

See Also

MGL_wmCreateWindow, *MGL_wmGetRootWindow*

MGL_wmGetWindowUserData

Reads user data of the window.

Declaration

```
void* MGLAPI MGL_wmGetWindowUserData(  
    window_t *wnd)
```

Prototype In

mgraph.h

Parameters

wnd the window

Return Value

User data pointer.

Description

Returns user data pointer of this window or NULL if you haven't called *MGL_wmSetWindowUserData* before.

See Also

MGL_wmSetWindowUserData

MGL_wmInvalidateRect

Invalidates rectangular part of device context.

Declaration

```
void MGLAPI MGL_wmInvalidateRect(  
    winmng_t *wm,  
    rect_t *rect)
```

Prototype In

mgraph.h

Parameters

<i>wm</i>	the window manager
<i>rect</i>	rectangular area to invalidate (in global coordinates)

Description

This function invalidates given rectangular area of device context that is associated with this window manager. The DC will be repainted next time you call *MGL_wmUpdateDC*.

See Also

MGL_wmUpdateDC, *MGL_wmInvalidateRegion*, *MGL_wmInvalidateWindow*,
MGL_wmInvalidateWindowRect, *MGL_wmInvalidateWindowRegion*

MGL_wmInvalidateRegion

Invalidates region of device context.

Declaration

```
void MGLAPI MGL_wmInvalidateRegion(  
    winmng_t *wm,  
    region_t *region)
```

Prototype In

mgraph.h

Parameters

<i>wm</i>	the window manager
<i>region</i>	region to invalidate (in global coordinates)

Description

This function invalidates given region of device context that is associated with this window manager. The DC will be repainted next time you call *MGL_wmUpdateDC*.

See Also

MGL_wmUpdateDC, *MGL_wmInvalidateRect*, *MGL_wmInvalidateWindow*,
MGL_wmInvalidateWindowRect, *MGL_wmInvalidateWindowRegion*

MGL_wmInvalidateWindow

Invalidates window.

Declaration

```
void MGLAPI MGL_wmInvalidateWindow(  
    window_t *wnd)
```

Prototype In

mgraph.h

Parameters

wnd the window to invalidate

Description

This function invalidates area of DC covered by the window.

The DC will be repainted next time you call *MGL_wmUpdateDC*.

See Also

MGL_wmUpdateDC, *MGL_wmInvalidateRect*, *MGL_wmInvalidateRegion*,
MGL_wmInvalidateWindowRect, *MGL_wmInvalidateWindowRegion*

MGL_wmInvalidateWindowRect

Invalidates rectangular area of window.

Declaration

```
void MGLAPI MGL_wmInvalidateWindowRect(  
    window_t *wnd,  
    rect_t *rect)
```

Prototype In

mgraph.h

Parameters

<i>wnd</i>	the window to invalidate
<i>rect</i>	rectangle to invalidate (in window's coordinates)

Description

This function invalidates given rectangular region of device context that is associated with this window manager. The rectangle is in coordinates relative to the window's upper left corner and is intersected with window's visible part before invalidating the DC.

The DC will be repainted next time you call *MGL_wmUpdateDC*.

See Also

MGL_wmUpdateDC, *MGL_wmInvalidateRect*, *MGL_wmInvalidateRegion*,
MGL_wmInvalidateWindow, *MGL_wmInvalidateWindowRegion*

MGL_wmInvalidateWindowRegion

Invalidates region of window.

Declaration

```
void MGLAPI MGL_wmInvalidateWindowRegion(
    window_t *wnd,
    region_t *region)
```

Prototype In

mgraph.h

Parameters

<i>wnd</i>	the window to invalidate
<i>region</i>	region to invalidate (in window's coordinates)

Description

This function invalidates given region of device context that is associated with this window manager. The region is in coordinates relative to the window's upper left corner and is intersected with window's visible part before invalidating the DC.

The DC will be repainted next time you call *MGL_wmUpdateDC*.

See Also

MGL_wmUpdateDC, *MGL_wmInvalidateRect*, *MGL_wmInvalidateRegion*,
MGL_wmInvalidateWindow, *MGL_wmInvalidateWindowRect*

MGL_wmLowerWindow

Lowers the window.

Declaration

```
void MGLAPI MGL_wmLowerWindow(  
    window_t *wnd)
```

Prototype In

mgraph.h

Description

Lowers the window so that it is behind all its siblings. The window does not cover any part of any of its siblings after this operation.

Note that if there are sibling windows with MGL_WM_STAY_ON_BOTTOM flag then the window will be place on top of them unless it itself has this flag.

See Also

MGL_wmSetWindowFlags, MGL_wmRaiseWindow

MGL_wmPopGlobalEventHandler

Removes top most global event handler.

Declaration

```
ibool MGLAPI MGL_wmPopGlobalEventHandler(  
    winmng_t *wm)
```

Prototype In

mgraph.h

Parameters

wm window manager

Return Value

False if the stack was already empty, true otherwise.

Description

This function removes the handler on the top of global event handlers stack, if there is any.

See Also

*MGL_wmProcessEvent, MGL_wmPushWindowEventHandler,
MGL_wmPopWindowEventHandler, MGL_wmRemoveWindowEventHandler,
MGL_wmPushGlobalEventHandler, MGL_wmRemoveGlobalEventHander,
MGL_wmCaptureEvents, MGL_wmUncaptureEvents*

MGL_wmPopWindowEventHandler

Removes top most event handler of the window.

Declaration

```
ibool MGLAPI MGL_wmPopWindowEventHandler(  
    window_t *wnd)
```

Prototype In

mgraph.h

Parameters

wnd window

Return Value

False if the stack was already empty, true otherwise.

Description

This function removes the handler on the top of handlers stack of the window, if there is any.

See Also

*MGL_wmProcessEvent, MGL_wmPushWindowEventHandler,
MGL_wmRemoveWindowEventHandler, MGL_wmPushGlobalEventHandler,
MGL_wmPopGlobalEventHandler, MGL_wmRemoveGlobalEventHander,
MGL_wmCaptureEvents, MGL_wmUncaptureEvents*

MGL_wmProcessEvent

Processes event and distributes it to windows.

Declaration

```
ibool MGLAPI MGL_wmProcessEvent (
    winmng_t *wm,
    event_t *event)
```

Prototype In

mgraph.h

Parameters

<i>wm</i>	window manager
<i>event</i>	event to distribute

Return Value

Returns true if the event was processed or false otherwise.

Description

MGL_wmProcessEvent distributes events to windows and passes them to event handlers that the user attached to windows. This happens in several steps:

First, global event handlers (see *MGL_wmPushGlobalEventHandler*) are searched for one that can handle this event (by ANDing event->what with the mask) and if such handler exists, the event is passed to it. If global handler returns true, processing ends and *MGL_wmProcessEvent* returns true. If the handler returns false, search continues among other global handlers.

Next, redirection table is searched for windows that have captured events of this type. If such window is found, it will be used in the next step. Otherwise, *MGL_wmProcessEvent* uses the window under mouse pointer.

Finally, event table of the window determined in the previous step is searched for a handler that accepts this type of events and the event is passed to it. If the handler returns true, the function returns with true, otherwise it continues with the rest of handlers.

If none of the above steps succeeded, false is returned.

Note: *This function also handles changes of mouse cursor as the pointer moves from one window to another.*

Note: *Mouse position information in event_t is NOT translated to window's local coordinates!*

Note: *A typical application will run in event loop and repeatedly call EVT_halt, MGL_wmProcessEvent and MGL_wmUpdateDC.*

See Also

*MGL_wmPushWindowEventHandler, MGL_wmPopWindowEventHandler,
MGL_wmRemoveWindowEventHandler, MGL_wmPushGlobalEventHandler,
MGL_wmPopGlobalEventHandler, MGL_wmRemoveGlobalEventHander,
MGL_wmCaptureEvents, MGL_wmUncaptureEvents*

MGL_wmPushGlobalEventHandler

Adds event handler to the window manager.

Declaration

```
void MGLAPI MGL_wmPushGlobalEventHandler(
    winmng_t *wm,
    globaleventhandler_t hndFunc,
    ulong mask,
    int id)
typedef ibool (MGLAPIP globaleventhandler_t)(event_t *event)
```

Prototype In mgraph.h

Parameters

<i>wm</i>	window manager
<i>hndFunc</i>	handler callback function
<i>mask</i>	mask of events the handler will handle (for <i>event_t.what</i>)
<i>id</i>	arbitrary user-choosen ID of this handler, used only when removing the event handler from the middle of handlers stack

Description

This function adds event handler to the stack of window manager's global event handlers. The meaning of parameters is same as in *MGL_wmPushWindowEventHandler* and they work in very similar way. The only difference is that global event handlers are independent of windows (they don't have window parameter) and that they take precedence over window-specific handlers.

See Also

MGL_wmProcessEvent, *MGL_wmPushWindowEventHandler*,
MGL_wmPopWindowEventHandler, *MGL_wmRemoveWindowEventHandler*,
MGL_wmPopGlobalEventHandler, *MGL_wmRemoveGlobalEventHandler*,
MGL_wmCaptureEvents, *MGL_wmUncaptureEvents*

MGL_wmPushWindowEventHandler

Adds event handler to the window.

Declaration

```
void MGLAPI MGL_wmPushWindowEventHandler(
    window_t *wnd,
    windoweventhandler_t hndFunc,
    ulong mask,
    int id)
typedef ibool (MGLAPIP windoweventhandler_t)(struct window_t
*wnd, event_t *event)
```

Prototype In

mgraph.h

Parameters

<i>wnd</i>	window
<i>hndFunc</i>	handler callback function
<i>mask</i>	mask of events the handler will handle (for <i>event_t.what</i>)
<i>id</i>	arbitrary user-choosen ID of this handler, used only when removing the event handler from the middle of handlers stack

Description

This function adds event handler to the stack of window's event handlers. These handlers are callback functions that are called by *MGL_wmProcessEvent* when it is determined that the event belongs to this window and event handler.

The algorithm used to determine target window for an event is described in *MGL_wmProcessEvent* documentation. The event is passed to the first event handler on the stack whose mask produces non-zero result of bitwise and with *event_t.what* member of the event.

An event handler returns true if it processed the event (that is, the event is processed no longer) or false if it didn't (in which case the event goes to the next handler on the stack).

Note: *Handlers that were added later take precedence over earlier pushed ones.*

See Also

MGL_wmProcessEvent, MGL_wmPopWindowEventHandler,
MGL_wmRemoveWindowEventHandler, MGL_wmPushGlobalEventHandler,
MGL_wmPopGlobalEventHandler, MGL_wmRemoveGlobalEventHandler,
MGL_wmCaptureEvents, MGL_wmUncaptureEvents

MGL_wmRaiseWindow

Raises the window.

Declaration

```
void MGLAPI MGL_wmRaiseWindow(  
    window_t *wnd)
```

Prototype In

mgraph.h

Description

Raises the window in front of its siblings. No part of the window is covered by any of its siblings after this operation.

See Also

MGL_wmSetWindowFlags, *MGL_wmLowerWindow*

MGL_wmRemoveGlobalEventHandler

Removes global event handler.

Declaration

```
ibool MGLAPI MGL_wmRemoveGlobalEventHandler(
    winmng_t *wm,
    int id)
```

Prototype In

mgraph.h

Parameters

wm window
id ID of the handler to remove

Return Value

False if the stack did not contain handler with this ID, true otherwise.

Description

This function removes the handler with given ID (as passed to MGL_wmPushWindowHandler) from global event handlers stack. There may not be any handler with such ID; in that case, the function returns false.

See Also

MGL_wmProcessEvent, MGL_wmPushWindowEventHandler,
MGL_wmPopWindowEventHandler, MGL_wmRemoveWindowEventHandler,
MGL_wmPushGlobalEventHandler, MGL_wmPopGlobalEventHandler,
MGL_wmCaptureEvents, MGL_wmUncaptureEvents

MGL_wmRemoveWindowEventHandler

Removes event handler of the window.

Declaration

```
ibool MGLAPI MGL_wmRemoveWindowEventHandler(
    window_t *wnd,
    int id)
```

Prototype In

mgraph.h

Parameters

<i>wnd</i>	window
<i>id</i>	ID of the handler to remove

Return Value

False if the stack did not contain handler with this ID, true otherwise.

Description

This function removes the handler with given ID (as passed to MGL_wmPushWindowHandler) from window's handlers stack. There may not be any handler with such ID; in that case, the function returns false.

See Also

MGL_wmProcessEvent, MGL_wmPushWindowEventHandler, MGL_wmPopWindowEventHandler, MGL_wmPushGlobalEventHandler, MGL_wmPopGlobalEventHandler, MGL_wmRemoveGlobalEventHandler, MGL_wmCaptureEvents, MGL_wmUncaptureEvents

MGL_wmReparentWindow

Reparents window.

Declaration

```
void MGLAPI MGL_wmReparentWindow(  
    window_t *wnd,  
    window_t *newParent)
```

Prototype In

mgraph.h

Parameters

<i>wnd</i>	the window
<i>newParent</i>	new parent

Description

Changes window's parent. This is two-stages operation: MGL removes the window from its current parent first and then adds it as a child of newParent.

See Also

MGL_wmCreateWindow, *MGL_wmGetWindowParent*

MGL_wmSetGlobalCursor

Sets global cursor.

Declaration

```
void MGLAPI MGL_wmSetGlobalCursor(
    winmng_t *wm,
    cursor_t *cursor)
```

Prototype In

mgraph.h

Parameters

<i>wm</i>	window manager
<i>cursor</i>	the cursor to set

Description

In MGL window manager, any window has associated mouse cursor that is shown whenever the pointer enters window's area. This function enables you to forcibly set one cursor for all windows (useful for e.g. hourglass cursor when performing a lengthy operation). MGL WM will ignore cursors associated with windows if there's a global one.

Call *MGL_wmSetGlobalCursor(NULL)* to disable global cursor.

Note: *MGL_wmSetGlobalCursor* does not take ownership of the cursor.

See Also

MGL_wmSetWindowCursor

MGL_wmSetWindowCursor

Sets mouse cursor specific to window.

Declaration

```
void MGLAPI MGL_wmSetWindowCursor(  
    window_t *wnd,  
    cursor_t *cursor)
```

Prototype In

mgraph.h

Parameters

<i>wnd</i>	window to set the cursor for
<i>cursor</i>	the cursor to set

Description

In MGL window manager, any window has associated mouse cursor that is shown whenever the pointer enters window's area. Call this function to set the cursor (default is arrow cursor).

Note: *MGL_wmSetWindowCursor does not take ownership of the cursor.*

See Also

MGL_wmSetGlobalCursor

MGL_wmSetWindowDestructor

Sets window's destructor callback.

Declaration

```
void MGLAPI MGL_wmSetWindowDestructor (
    window_t *wnd,
    windtor_t dtor)
typedef void (MGLAPIP windtor_t) (struct window_t *wnd)
```

Prototype In

mgraph.h

Parameters

<i>wnd</i>	the window
<i>dtor</i>	the destructor callback

Description

Use this function to set window's destructor callback. The destructor is called by *MGL_wmDestroyWindow* before destroying children and deallocating *window_t* structure. This function gives you a chance to react to window destruction (for example by freeing data set with *MGL_wmSetWindowUserData*).

Note: Remember that you don't always destroy windows with *MGL_wmDestroyWindow*; some windows are destroyed "implicitly". This happens when you call *MGL_wmDestroyWindow* on window that itself has child windows. Children are recursively destroyed and destructor callback is the only way to notify your code about it. As a special case, this also happens in *MGL_wmDestroy* because it calls *MGL_wmDestroyWindow* on the root window.

See Also

MGL_wmDestroyWindow, *MGL_wmSetWindowUserData*

MGL_wmSetWindowFlags

Sets flags affecting window's behavior.

Declaration

```
void MGLAPI MGL_wmSetWindowFlags(  
    window_t *wnd,  
    long flags)
```

Prototype In

mgraph.h

Parameters

<i>wnd</i>	the window
<i>flags</i>	combination of flags defined in <i>MGL_wmWindowFlags</i> enum

Description

This function is usually called immediately after creating a window to define its behavior. Flags may be 0 (the default) or any or-combination of possible flags.

This function may be called several times, MGL WM will update the window as necessary (for example, you may call *MGL_wmSetWindowFlags(wnd, MGL_WM_ALWAYS_ON_TOP)* and later *MGL_wmSetWindowFlags(wnd, 0)* to disable the behavior).

See Also

MGL_wmGetWindowFlags, *MGL_wmCreateWindow*

MGL_wmSetWindowPainter

Sets painter callback for the window.

Declaration

```
void MGLAPI MGL_wmSetWindowPainter(
    window_t *wnd,
    painter_t painter)
typedef void (MGLAPI painter_t)(struct window_t *wnd, MGLDC *dc)
```

Prototype In mgraph.h

Parameters

<i>wnd</i>	the window
<i>painter</i>	painter callback

Description

The painter is responsible for redrawing the window it is attached to. It can only draw to the part of device context covered by the window.

Every window must have a painter callback associated with it. MGL WM calls painters as necessary from *MGL_wmUpdateDC* to ensure that the device context reflects current status of windows.

The painter callback takes two arguments: one of them is the window being painted and the other is the device context to paint on. The DC is already made current and clipping region is properly set.

Note: You can use *MGL_wmSetWindowUserData* to attach arbitrary data to the window and use this data in painter function.

See Also

MGL_wmSetWindowFlags, *MGL_wmShowWindow*, *MGL_wmCreateWindow*,
MGL_wmUpdateDC, *MGL_wmBeginPaint*, *MGL_wmSetWindowUserData*,
MGL_wmInvalidateWindow

MGL_wmSetWindowPosition

Moves the window to new position and/or changes its size.

Declaration

```
void MGLAPI MGL_wmSetWindowPosition(
    window_t *wnd,
    int x,
    int y,
    size_t width,
    size_t height)
```

Prototype In mgraph.h

Parameters

<i>wnd</i>	the window
<i>x</i>	new X coordinate of window position
<i>y</i>	new Y coordinate of window position
<i>width</i>	new window width
<i>height</i>	new window height

Description

Changes window's position and size to new values and repaints the DC as necessary. Depending on window's flags and status, different actions are triggered by the call to *MGL_wmSetWindowPosition*:

1. If the new size and position are same as old ones, nothing happens. 2. If *wnd* is top level window (i.e. its parent is *wnd->wm->rootWnd*) and size is unmodified and only position changes, optimized routine is used and the window is copied to its new position with *MGL_bitBlt*. 3. If the window has *MGL_WM_FULL_REPAINT_ON_RESIZE* flag or its position (i.e. not only size) changed, it is completely invalidated and queued for repaint. 4. Otherwise (i.e. the window doesn't have *MGL_WM_FULL_REPAINT_ON_RESIZE* and it only changed its size), only the different between old and new window area is repainted.

Note: *The change in position won't be visible until the next call to MGL_wmUpdateDC. As an exception, if optimized bitblt move was used, the change is visible immediately.*

See Also

MGL_wmSetWindowFlags, *MGL_wmUpdateDC*, *MBL_bitBlt*

MGL_wmSetWindowUserData

Associates user data with the window.

Declaration

```
void MGLAPI MGL_wmSetWindowUserData(  
    window_t *wnd,  
    void *data)
```

Prototype In

mgraph.h

Parameters

<i>wnd</i>	the window
<i>data</i>	pointer to user data

Description

This function is primarily meant for interaction with higher-level windowing system built on top of MGL WM. *window_t* structure contains user data pointer that is not used for anything by MGL, but the user is provided with functions for reading and writing it (*MGL_wmSetWindowUserData* and *MGL_wmGetWindowUserData*).

You probably want to set window destructor, too, in order to free user data before the window is destroyed.

See Also

MGL_wmGetWindowUserData, *MGL_wmSetWindowDestructor*

MGL_wmShowWindow

Shows or hides the window

Declaration

```
void MGLAPI MGL_wmShowWindow(  
    window_t *wnd,  
    ibool show)
```

Prototype In

mgraph.h

Parameters

<i>wnd</i>	the window
<i>show</i>	true if the window shall be shown, false if hidden

Description

If show is true, shows the window and invalidates device context appropriately. If show is false, hides the window.

Hidden window is not visible, does not accept events and *MGL_wmGetWindowAtPosition* will never test against it. It is still a valid window, though, and you can manipulate it just like any other window.

MGL_wmUncaptureEvents

Captures events and redirects them to specific window.

Declaration

```
void MGLAPI MGL_wmUncaptureEvents (
    window_t *wnd,
    int id)
```

Prototype In

mgraph.h

Parameters

<i>wnd</i>	window
<i>id</i>	ID of the capture record to remove

Description

Cancels the instruction to capture events to the window as previously specified by a call to *MGL_wmCaptureEvents*.

See Also

MGL_wmProcessEvent, *MGL_wmPushWindowEventHandler*,
MGL_wmPopWindowEventHandler, *MGL_wmRemoveWindowEventHandler*,
MGL_wmPushGlobalEventHandler, *MGL_wmPopGlobalEventHandler*,
MGL_wmRemoveGlobalEventHandler, *MGL_wmCaptureEvents*

MGL_wmUpdateDC

Updates invalidated parts of window manager's DC.

Declaration

```
void MGLAPI MGL_wmUpdateDC (
    winmng_t *wm)
```

Prototype In

mgraph.h

Parameters

wm the window manager

Description

This function updates the device context by redrawing all areas previously invalidated by either explicit calls to MGL_wmInvalidateXXX functions or by operations that modify visual appearance of windows (such as MGL_wmSetWindowPosition, MGL_wmShowWindow or MGL_wmRaiseWindow).

Painter callbacks attached to windows are used to do actual painting.

Note: *A typical application will run in event loop and repeatedly call EVT_halt, MGL_wmProcessEvent and MGL_wmUpdateDC.*

Note: *This function automatically hides mouse pointer if neccessary to avoid occurence rendering artifacts.*

See Also

MGL_wmInvalidateRect, MGL_wmInvalidateRegion, MGL_wmInvalidateWindow, MGL_wmInvalidateWindowRect, MGL_wmInvalidateWindowRegion, MGL_wmProcessEvent, MGL_wmSetWindowPainter

MS_getPos

Returns the current mouse cursor location.

Declaration

```
void MGLAPI MS_getPos (  
    int *x,  
    int *y)
```

Prototype In

mgraph.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

MS_moveTo

MS_hide

Hides the mouse cursor.

Declaration

```
void MGLAPI MS_hide(void)
```

Prototype In

mgraph.h

Description

Decrements the internal mouse cursor display counter, and hides the cursor if the counter was previously set to zero. Calls to *MS_show* increment the counter, allowing the *MS_show* and *MS_hide* calls to be nested.

See Also

MS_show, *MS_obscure*

MS_moveTo

Moves the mouse cursor to a new location.

Declaration

```
void MGLAPI MS_moveTo(  
    int x,  
    int y)
```

Prototype In

mgraph.h

Parameters

<i>x</i>	New mouse x coordinate (screen coordinates)
<i>y</i>	New mouse y coordinate (screen coordinates)

Description

Moves the mouse cursor to the specified location in screen coordinates.

Note that it is not usually a good idea to move the mouse cursor around while the user is interacting with the application, but this can be used to restore the mouse cursor to a known location if it has been hidden for a long period of time.

See Also

MS_getPos

MS_obscure

Hides the mouse cursor from view during graphics output.

Declaration

```
void MGLAPI MS_obscure(void)
```

Prototype In

mgraph.h

Description

Hides the mouse cursor from view in order to perform graphics output using MGL. If the graphics device driver supports a hardware cursor, this is handled by the hardware, otherwise it is removed from the display. You should call this routine rather than *MS_hide* in order to temporarily hide the mouse cursor during graphics output as the *MS_hide* routine will always hide the cursor, regardless of whether the system has a hardware mouse cursor or not.

See Also

MS_show, *MS_hide*

MS_setCursor

Sets the mouse cursor shape.

Declaration

```
void MGLAPI MS_setCursor(  
    cursor_t *curs)
```

Prototype In

mgraph.h

Parameters

curs Pointer to new mouse cursor shape

Description

Sets the graphics mouse cursor shape, passed in the *cursor_t* structure. The *cursor_t* structure contains a mouse cursor AND mask and a mouse cursor XOR mask that is used to display the cursor on the screen, along with the mouse cursor hotspot location. Refer to the *cursor_t* structure definition for more information.

MS_setCursorColor

Sets the current mouse cursor color.

Declaration

```
void MGLAPI MS_setCursorColor(  
    color_t color)
```

Prototype In

mgraph.h

Parameters

color New mouse cursor color, in current display mode format.

Description

Sets the color for the mouse cursor to the specified color, which is passed in as a packed MGL color in the proper format for the current display mode (either a color index or a packed RGB color value). By default the mouse cursor is set to white, which is a color index of 15 by default in MGL. If you re-program the color palette in 4 or 8 bit modes, you will need to reset the mouse cursor value to the value that represents white.

MS_setCursorColorExt

Sets the current mouse cursor color.

Declaration

```
void MGLAPI MS_setCursorColorExt(  
    color_t foreColor,  
    color_t backColor)
```

Prototype In

mgraph.h

Parameters

<i>foreColor</i>	Mouse foreground cursor color, in current display mode format.
<i>backColor</i>	Mouse background cursor color, in current display mode format.

Description

Sets the colors for the mouse cursor to the specified color, which are passed in as a packed MGL colors in the proper format for the current display mode (either a color index or a packed RGB color value). By default the mouse cursor is set to white on black, which is a color index of 15 by default in MGL. If you re-program the color palette in 4 or 8 bit modes, you will need to reset the mouse cursor value to the value that represents white.

MS_show

Displays the mouse cursor.

Declaration

```
void MGLAPI MS_show(void)
```

Prototype In

mgraph.h

Description

Increments the internal mouse cursor display counter, and displays the cursor when the counter gets to zero. Calls to *MS_hide* decrement the counter, and this call effectively cancels a single *MS_hide* call, allowing the *MS_show* and *MS_hide* calls to be nested.

If the mouse was obscured with the *MS_obscure* function, this reverses the effect and will redisplay the mouse cursor again. On systems with a hardware mouse cursor, the *MS_obscure* function effectively does nothing, while on systems using a software mouse cursor, the *MS_obscure* function simply calls *MS_hide*.

Note that the mouse cursor display counter is reset to -1 by default when an MGL fullscreen mode is started, so a single *MS_show* will display the mouse cursor after the mode has been started.

See Also

MS_hide, *MS_obscure*

SPR_destroyBitmap

Destroy a bitmap managed by the sprite manager

Declaration

```
void MGLAPI SPR_destroyBitmap(  
    SPR_bitmapManager *mgr,  
    SPR_bitmap *bmp)
```

Prototype In

gm/sprite.h

Parameters

<i>mgr</i>	Bitmap manager to destroy sprite in
<i>bmp</i>	MGL bitmap to destroy

Return Value

Pointer to the loaded sprite object

Description

This function removes a bitmap to the Sprite Manager.

See Also

SPR_mgrAddOpaqueBitmap, SPR_mgrAddTransparentBitmap

SPR_draw

Draws the sprite object at the specified location.

Declaration

```
void MGLAPI SPR_draw(  
    SPR_bitmap *bmp,  
    int x,  
    int y)
```

Prototype In

gm/sprite.h

Parameters

<i>bmp</i>	Sprite object to draw
<i>x</i>	X coordinate to draw the sprite at
<i>y</i>	Y coordinate to draw the sprite at

Description

This function draws the sprite object at the specified (x,y) location on the device context currently bound to the Sprite Manager (ie: the device context you used when you initialized it with *SPR_mgrInit*). The sprite is drawn using the attributes of the sprite when you added it (ie: opaque or source transparent).

See Also

SRP_drawSection, *SPR_drawExt*, *SPR_mgrAddOpaqueBitmap*,
SPR_mgrAddTransparentBitmap,

SPR_drawExt

Draws the sprite object at the specified location.

Declaration

```
void MGLAPI SPR_drawExt(  
    SPR_bitmap *bmp,  
    int x,  
    int y,  
    int op)
```

Prototype In

gm/sprite.h

Parameters

<i>bmp</i>	Sprite object to draw
<i>x</i>	X coordinate to draw the sprite at
<i>y</i>	Y coordinate to draw the sprite at
<i>op</i>	Write mode to draw with

Description

This function draws the sprite object at the specified (x,y) location on the device context currently bound to the Sprite Manager (ie: the device context you used when you initialized it with *SPR_mgrInit*). The sprite is drawn using the attributes of the sprite when you added it (ie: opaque or source transparent).

See Also

SRP_drawSection, *SPR_draw*, *SPR_mgrAddOpaqueBitmap*,
SPR_mgrAddTransparentBitmap,

SPR_drawSection

Draws a section of a sprite object at the specified location.

Declaration

```
void MGLAPI SPR_drawSection(
    SPR_bitmap *bmp,
    int left,
    int top,
    int right,
    int bottom,
    int dstLeft,
    int dstTop)
```

Prototype In

gm/sprite.h

Parameters

<i>bmp</i>	Sprite object to draw
<i>left</i>	Left coordinate of area to copy from bitmap
<i>top</i>	Top coordinate of area to copy from bitmap
<i>right</i>	Right coordinate of area to copy from bitmap
<i>bottom</i>	Bottom coordinate of area to copy from bitmap
<i>dstLeft</i>	X coordinate to draw bitmap at
<i>dstTop</i>	Y coordinate to draw bitmap at

Description

This function draws the sprite object at the specified (x,y) location on the device context currently bound to the Sprite Manager (ie: the device context you used when you initialized it with *SPR_mgrInit*). The sprite is drawn using the attributes of the sprite when you added it (ie: opaque or source transparent).

See Also

SRP_draw, *SPR_drawExt*, *SPR_mgrAddOpaqueBitmap*, *SPR_mgrAddTransparentBitmap*,

SPR_drawSectionExt

Draws a section of a sprite object at the specified location.

Declaration

```
void MGLAPI SPR_drawSectionExt(
    SPR_bitmap *bmp,
    int left,
    int top,
    int right,
    int bottom,
    int dstLeft,
    int dstTop,
    int op)
```

Prototype In

gm/sprite.h

Parameters

<i>bmp</i>	Sprite object to draw
<i>left</i>	Left coordinate of area to copy from bitmap
<i>top</i>	Top coordinate of area to copy from bitmap
<i>right</i>	Right coordinate of area to copy from bitmap
<i>bottom</i>	Bottom coordinate of area to copy from bitmap
<i>dstLeft</i>	X coordinate to draw bitmap at
<i>dstTop</i>	Y coordinate to draw bitmap at
<i>op</i>	Write mode to draw with

Description

This function draws the sprite object at the specified (x,y) location on the device context currently bound to the Sprite Manager (ie: the device context you used when you initialized it with *SPR_mgrInit*). The sprite is drawn using the attributes of the sprite when you added it (ie: opaque or source transparent).

See Also

SRP_draw, *SPR_drawExt*, *SPR_mgrAddOpaqueBitmap*, *SPR_mgrAddTransparentBitmap*,

SPR_mgrAddOpaqueBitmap

Adds an opaque or non-transparent bitmap to the Sprite Manager

Declaration

```
SPR_bitmap * MGLAPI SPR_mgrAddOpaqueBitmap(
    SPR_bitmapManager *mgr,
    bitmap_t *bmp)
```

Prototype In

gm/sprite.h

Parameters

mgr Bitmap manager to add sprite to
bmp MGL bitmap to add to the Sprite Manager

Return Value

Pointer to the loaded sprite object

Description

This function adds a new opaque or non-transparent bitmap to the Sprite Manager. When you add the bitmap, the Sprite Manager takes over ownership of the memory allocated to the bitmap and you *must* not call *MGL_unloadBitmap* on the bitmap to free the memory. Once the bitmap has been added, you can then draw the bitmap by calling *SPR_draw* and pass in the pointer to the bitmap returned by this function.

Note: *The bitmap added to the Sprite Manager must be in the same format at the device context that the bitmaps will be copied to. Hence you should add code to your program to do any necessary conversions to the destination pixel format for the bitmaps (see the Fox & Bear sample program which contains code to do this).*

Note: *The Sprite Manager also maintains ownership of the SPR_bitmap object that is returned by this function, and this object will be destroyed automatically when you empty or exit the Sprite Manager. You can call SPR_destroyBitmap if you wish to destroy the bitmap.*

See Also

SPR_mgrAddTransparentBitmap, SPR_draw, SPR_mgrEmpty, SPR_destroyBitmap

SPR_mgrAddTransparentBitmap

Adds a source transparent bitmap to the Sprite Manager

Declaration

```
SPR_bitmap * MGLAPI SPR_mgrAddTransparentBitmap(
    SPR_bitmapManager *mgr,
    bitmap_t *bmp,
    color_t transparent)
```

Prototype In

gm/sprite.h

Parameters

<i>mgr</i>	Bitmap manager to add sprite to
<i>bmp</i>	MGL bitmap to add to the Sprite Manager
<i>transparent</i>	Transparent color for the bitmap

Return Value

Pointer to the loaded sprite object

Description

This function adds a new transparent bitmap to the Sprite Manager. When you add the bitmap, the Sprite Manager takes over ownership of the memory allocated to the bitmap and you *must* not call *MGL_unloadBitmap* on the bitmap to free the memory. Once the bitmap has been added, you can then draw the bitmap by calling *SPR_draw* and pass in the pointer to the bitmap returned by this function.

Note that the transparent color you pass in is a *source transparent* color, which means that pixels in the source bitmap that match the transparent color will not be drawn when you call *SPR_draw* for the returned sprite object (ie: they are transparent).

Note: *The bitmap added to the Sprite Manager must be in the same format at the device context that the bitmaps will be copied to. Hence you should add code to your program to do any necessary conversions to the destination pixel format for the bitmaps (see the Fox & Bear sample program which contains code to do this).*

Note: *The Sprite Manager also maintains ownership of the SPR_bitmap object that is returned by this function, and this object will be destroyed automatically when you empty or exit the Sprite Manager. You can call SPR_destroyBitmap if you wish to destroy the bitmap.*

See Also

SPR_mgrAddOpaqueBitmap, SPR_draw, SPR_mgrEmpty, SPR_destroyBitmap

SPR_mgrEmpty

Empties Sprite Manager of all loaded bitmaps

Declaration

```
void MGLAPI SPR_mgrEmpty(  
    SPR_bitmapManager *mgr)
```

Prototype In

gm/sprite.h

Parameters

mgr Bitmap manager to empty

Description

This function empties the Sprite Manager of all currently loaded bitmaps and deallocates the memory owned by those bitmaps. This function is most useful to clearing the Sprite Manager of all bitmaps when moving from one level to another in your game.

See Also

SPR_mgrInit, SPR_mgrEmpty

SPR_mgrExit

Shuts down the Game Framework Sprite Manager

Declaration

```
void MGLAPI SPR_mgrExit(  
    SPR_bitmapManager *mgr)
```

Prototype In

gm/sprite.h

Parameters

mgr Bitmap manager to shut down

Description

This function shuts down the Sprite Manager, and destroys all bitmaps currently own by the Sprite Manager.

See Also

SPR_mgrInit, *SPR_mgrEmpty*

SPR_mgrInit

Initializes the Game Framework Sprite Manager

Declaration

```
SPR_bitmapManager * MGLAPI SPR_mgrInit(  
    MGLDC *dc)
```

Prototype In

gm/sprite.h

Parameters

dc MGL device context to use

Return Value

Pointer to the bitmap manager for the device context.

Description

This function initializes the Sprite Manager library and sets up for storing bitmaps with the sprite manager. The device context you pass in can be any MGL device context, but usually it should be an MGL display device context or an MGL memory device context. The Sprite Manager will automatically interrogate the capabilities of the device context, and if the device context supports the creation of an offscreen device context for storing sprites in video memory, an offscreen device context will be created and managed by the sprite manager.

The sprite manager automatically does all the work to keep track of which bitmaps are loaded in video memory and which are loaded in system memory. If there is not enough video memory left to store a sprite when you add it to the Sprite Manager, that sprite is stored in system memory automatically.

Note: *When you add bitmaps to the Sprite Manager, those bitmaps are copied into the allocated buffer, so you should destroy the bitmaps when you are done. You can also optionally add bitmaps from another device context, which allows you to use a common device context to do conversion of bitmaps to the hardware color depth.*

Note: *If you switch fullscreen graphics modes or you switch from fullscreen modes to windowed modes, you must call SPR_mgrExit to destroy the bitmap manager and re-initialize it with the newly created device contexts. This is necessary because the capabilities of the new device context and the amount of available offscreen video memory (if any) will be different in the new graphics mode and all the sprites will have to be re-loaded into the Sprite Manager.*

See Also

SPR_mgrExit

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).

demo**Declaration**

```
void demo (MGLDC *dc)
```

writeCursor**Declaration**

```
void writeCursor(FILE *f, char *cName, char *name, char *maskName)
```

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

<i>EVT_LEFTBUT</i>	Set if left mouse button was down
<i>EVT_RIGHTBUT</i>	Set if right mouse button was down
<i>EVT_MIDDLEBUT</i>	Set if the middle button was down
<i>EVT_RIGHTSHIFT</i>	Set if right shift was down
<i>EVT_LEFTSHIFT</i>	Set if left shift was down
<i>EVT_RIGHTCTRL</i>	Set if right ctrl key was down
<i>EVT_RIGHTALT</i>	Set if right alt key was down
<i>EVT_LEFTCTRL</i>	Set if left ctrl key was down
<i>EVT_LEFTALT</i>	Set if left alt key was down
<i>EVT_SHIFTKEY</i>	Mask for any shift key down
<i>EVT_CTRLSTATE</i>	Set if ctrl key was down
<i>EVT_ALTSTATE</i>	Set if alt key was down
<i>EVT_CAPSLOCK</i>	Caps lock is active
<i>EVT_NUMLOCK</i>	Num lock is active
<i>EVT_SCROLLLOCK</i>	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.*

GMDC

Declaration

```
typedef struct {
    MGLDC          *dc;
    MGLDC          *dispdc;
    MGLDC          *backdc;
    int             numModes;
    int             numFullscreenModes;
    ulong          modeFlags;
    GM_modeInfo     modeList[GM_MAXMODE+1];
#ifdef __WINDOWS__ && !defined(__CONSOLE__)
    MGL_HWND        mainWindow;
#endif
} GMDC
```

Prototype In

gm\gm.h

Description

Main structure for maintaining the state information for the Game Framework. The application program always does all drawing and rendering to the *GMDC* dc member, which will draw directly to the framebuffer or to a system memory buffer depending on the hardware and the initialization information. The modeFlags field contains the original mode flags information passed to *GM_init*, which defines which color depths your game will support. The modeList contains a complete list of all the available graphics modes supported by the Game Framework, including *psuedo* modes that are modes that include auto-stretching.

The *dispdc* and *backdc* field are primarily for internal use by the Game Framework, and you should not use those fields unless you are clear what they are used for.

Members

<i>dc</i>	DC for drawing
<i>dispdc</i>	Main display DC
<i>backdc</i>	Back buffer if necessary (could be system or video memory)
<i>numModes</i>	Number of modes in the mode list
<i>numFullscreenModes</i>	Number of fullscreen capable modes in the mode list
<i>modeFlags</i>	Mode flags for current graphics mode
<i>modeList</i>	List of all available modes supported by the Game Framework
<i>mainWindow</i>	Handle to main window (Windows only)

GM_driverOptions

Declaration

```
typedef struct {
    ibool                useSNAP;
    ibool                useHWOpenGL;
    MGL_glOpenGLType    openGLType;
    GM_modeFlagsType    modeFlags;
} GM_driverOptions
```

Prototype In

gm\gm.h

Description

This structure contains the flags for letting the Game Framework know which drivers should be registered with the MGL to enable support for different device driver technologies. By default all these drivers are enabled, and you can change the values of these flags by calling *GM_setDriverOptions* before calling *GM_init*.

Members

<i>useSNAP</i>	True to enable SciTech SNAP Graphics support
<i>useHWOpenGL</i>	True to enable OpenGL hardware support
<i>openGLType</i>	OpenGL type to be used (defaults to MGL_GL_AUTO)
<i>modeFlags</i>	Mode flags for supported color depths

GM_modeFlagsType

Declaration

```
typedef enum {
    GM_MODE_8BPP          = 0x01,
    GM_MODE_16BPP         = 0x02,
    GM_MODE_24BPP         = 0x04,
    GM_MODE_32BPP         = 0x08,
    GM_ONLY_2D_ACCEL      = 0x10,
    GM_ONLY_3D_ACCEL      = 0x20,
    GM_ALLOW_WINDOWED     = 0x40,
    GM_MODE_ALLBPP        = 0x0F
} GM_modeFlagsType
```

Prototype In

gm\gm.h

Description

Mode flags to pass to *GM_init*. These flags inform the Game Framework which color depths you want to support in your game, and can be a logical OR combination of all the available flags. Hence if you game only supports 8bpp modes then you would pass in *GM_MODE_8BPP*. If you game only supports 8bpp and 16bpp, then you would pass in *GM_MODE_8BPP | GM_MODE_16BPP*.

Note: *GM_MODE_16BPP* includes support for both 15bpp (5:5:5) and 16bpp (5:6:5) MGL modes and if you support *GM_MODE_16BPP* then you will have to be able to support either format on the end users system.

Members

<i>GM_MODE_8BPP</i>	Include support for 8bpp modes
<i>GM_MODE_16BPP</i>	Include support for 15bpp and 16bpp modes
<i>GM_MODE_24BPP</i>	Include support for 24bpp modes (3 bytes per pixel)
<i>GM_MODE_32BPP</i>	Include support for 32bpp modes (4 bytes per pixel)
<i>GM_ONLY_2D_ACCEL</i>	Only include modes with 2D hardware support
<i>GM_ONLY_3D_ACCEL</i>	Only include modes with 3D hardware support
<i>GM_ALLOW_WINDOWED</i>	Allow for windowed only modes in mode list
<i>GM_MODE_ALLBPP</i>	Include support for all color depths

GM_modeInfo

Declaration

```
typedef struct {
    int          xRes;
    int          yRes;
    int          bitsPerPixel;
    int          mode;
    int          pages;
    unsigned long flags;
    char         driverName[60];
    GM_stretchType stretch;
    GM_stretchType windowedStretch;
} GM_modeInfo
```

Prototype In

gm\gm.h

Description

The structure maintains information about the graphics modes that are supported by the game framework and is passed to *GM_setMode* to specify the mode to be initialized.

Note that the xRes and yRes values are the logical resolution for the mode which may be different to the physical resolution, since the Game Framework also enumerates *pseudo* modes that use stretching. Hence even if the hardware does not have native support for a 320x240 mode, it may appear in the list using 320x480 as the real mode and a stretch factor of 1x2 or using 640x480 as the real mode and a stretch factor of 2x2.

If you wish to set a windowed mode directly set the mode parameter to grWINDOWED and the mode will start as a windowed mode.

Members

<i>xRes</i>	Logical X resolution for mode (not physical!)
<i>yRes</i>	Logical Y resolution for mode (not physical!)
<i>bitsPerPixel</i>	Color depth for mode. Note 16bpp includes 15bpp (5:5:5)
<i>mode</i>	Fullscreen MGL mode number (-1 means windowed mode)
<i>pages</i>	Number of hardware display pages for mode
<i>flags</i>	Mode flags for the mode
<i>driverName</i>	Name of driver that will be used in fullscreen modes
<i>stretch</i>	Stretch factor for the mode
<i>windowedStretch</i>	Stretch factor to use in windowed modes

GM_stretchType

Declaration

```
typedef enum {  
    GM_STRETCH_1x1,  
    GM_STRETCH_1x2,  
    GM_STRETCH_2x2  
} GM_stretchType
```

Prototype In

gm\gm.h

Description

Stretch flags to pass to *GM_setMode*. The 1x2 stretch is useful for rendering in 320x200/240 modes on hardware that can't do double scanning but does support 320x400/480 modes natively. 1x2 stretching in the MGL is very efficient so this is better than rendering at 320x400 natively if each pixel takes a lot of computation time.

Members

<i>GM_STRETCH_1x1</i>	No stretching
<i>GM_STRETCH_1x2</i>	1x2 stretch (2x vertical stretch)
<i>GM_STRETCH_2x2</i>	2x2 stretch (2x stretch in both X and Y)

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

MGLBUF

Declaration

```
typedef struct {
    M_uint32    dwSize;
    M_int32     width;
    M_int32     height;
    M_int32     bytesPerLine;
    M_int32     cacheBytesPerLine;
    M_int32     startX;
    M_int32     startY;
    M_int32     offset;
    M_int32     flags;
    M_int32     format;
    M_int32     usageCount;
    void        *surface;
    void        *surfaceCache;
    MGLDC       *dc;
} MGLBUF
```

Prototype In

mgraph.h

Description

Structure representing a lightweight bitmap buffer in offscreen display memory. Offscreen buffers are used to store bitmap and sprite information in the offscreen memory in the hardware, but are not full offscreen DC's and hence have much less memory overhead than a full offscreen DC. Buffers can only be used for storing bitmaps and blitting the around on the screen. You can copy the contents to a MGL device context using the *MGL_putBuffer*, *MGL_stretchBuffer* and *MGL_putBufferTransparent* functions. You can also copy the contents of an MGL device context to a buffer using the *MGL_copyToBuffer* function.

If you need to draw on a buffer in offscreen memory, create a full offscreen device context instead, and then you can call any of the standard MGL drawing functions and BitBlt operations for the offscreen memory buffer. The primary disadvantage of doing this is that a full offscreen device context has a lot more memory overhead involved in maintaining the device context state information than a simple offscreen buffer.

Note: *The MGL automatically manages offscreen display memory, and if you run out of offscreen display memory it will place the buffer surfaces in system memory. Hence you should allocate your important buffers first, to ensure they end up in offscreen memory.*

Note: *The MGL also manages the surface memory for the buffers, so that if your offscreen surfaces get lost (ie: on a fullscreen mode switch), they will automatically be restored when the application regains the active focus. This also allows the MGL to do automatic offscreen heap compaction when necessary to free up empty space on the heap.*

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>	Size of the structure in bytes
<i>width</i>	Width of the offscreen buffer in pixels
<i>height</i>	Height of the offscreen buffer in pixels
<i>bytesPerLine</i>	Scanline width for the buffer in bytes
<i>cacheBytesPerLine</i>	Scanline width for the system memory buffer cache
<i>startX</i>	Internal value; do not use
<i>startY</i>	Internal value; do not use
<i>offset</i>	Internal value; do not use
<i>flags</i>	Flags for the buffer (<i>MGL_bufferFlagsType</i>)
<i>format</i>	Internal value; do not use
<i>usageCount</i>	Useage 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 uncached)
<i>dc</i>	Display device context buffer is associated with

MGLDC

Declaration

```
typedef struct {
    attributes_t    a;
    void            *surface;
    gmode_t         mi;
    pixel_format_t  pf;
    color_t         *colorTab;
    void            *userData;
} MGLDC
```

Prototype In mgraph.h

Description

Structure representing the public structure of all MGL device contexts. You can use the information in this structure to directly access the device surface for developing your own custom rendering code with the MGL.

Note: *The size of an MGLDC structure is a lot bigger internally that is declared in this header file, as we only expose the core information necessary for application programs that does not change from release to release. Internally the MGL deals with an expanded structure that contains all the information.*

Members

<i>a</i>	Current device attributes
<i>surface</i>	Pointer to device surface. This pointer will be a pointer to the start of framebuffer memory for the device context.
<i>mi</i>	Mode attribute information for the device
<i>pf</i>	Current pixel format for the device context.
<i>colorTab</i>	Color lookup table cache for the device context. In color index modes this is an array of 256 <i>palette_t</i> structures that represent the color palette for the device context. In 15-bits per pixel and higher modes, this is an array of 256 <i>color_t</i> values which contains a lookup table with pre-packed color values for the current display mode. This is used by the MGL when translating color index bitmaps and drawing them in RGB display modes.
<i>userData</i>	Arbitrary pointer to user defined data. This allows the application programmer to store their own user defined data in the <i>MGLDC</i> structure, or via an object attached to the <i>MGLDC</i> structure.

MGLVisual

Declaration

```
typedef struct {
    ibool    rgb_flag;
    ibool    alpha_flag;
    ibool    db_flag;
    int      depth_size;
    int      stencil_size;
    int      accum_size;
    ibool    stereo_flag;
} MGLVisual
```

Prototype In

mgraph.h

Description

Structure representing the information passed to the MGL's OpenGL binding functions to both choose an appropriate visual that is supported by the OpenGL implementation and to pass in the information for the visual when a rendering context is created.

Application code will fill in the structure and call *MGL_glChooseVisual* to find out a valid set of visual information that is close to what they requested, then call *MGL_glSetVisual* to make that the current visual for a specific MGL device context. The the next call to *MGL_glCreateContext* will use the visual information previously requested with the call to *MGL_glSetVisual*.

Members

<i>rgb_flag</i>	True for an RGB mode, false for color index modes
<i>alpha_flag</i>	True for alpha buffers (8-bits deep)
<i>db_flag</i>	True for double buffered, false for single buffered
<i>depth_size</i>	Size of depth buffer in bits
<i>stencil_size</i>	Size of stencil buffer in bits
<i>accum_size</i>	Size of accumulation buffer in bits
<i>stereo_flag</i>	True for a stereo display false

MGL_COLORS

Declaration

```
typedef enum {  
    MGL_BLACK,  
    MGL_BLUE,  
    MGL_GREEN,  
    MGL_CYAN,  
    MGL_RED,  
    MGL_MAGENTA,  
    MGL_BROWN,  
    MGL_LIGHTGRAY,  
    MGL_DARKGRAY,  
    MGL_LIGHTBLUE,  
    MGL_LIGHTGREEN,  
    MGL_LIGHTCYAN,  
    MGL_LIGHTRED,  
    MGL_LIGHTMAGENTA,  
    MGL_YELLOW,  
    MGL_WHITE  
} MGL_COLORS
```

Prototype In

mgraph.h

Description

Defines the MGL standard colors. This is the standard set of colors for the IBM PC in DOS graphics modes. The default palette will have been programmed to contain these values by the MGL when a graphics mode is started. If the palette has been changed, the colors on the screen will not correspond to the names defined here. Under a Windowing system (like Windows, OS/2 PM or X Windows) these colors will not correspond to the default colors. For Windows see the *MGL_WIN_COLORS* enumeration which defines the default colors in a windowed mode.

MGL_WIN_COLORS

Declaration

```
typedef enum {
    MGL_WIN_BLACK           = 0,
    MGL_WIN_DARKRED         = 1,
    MGL_WIN_DARKGREEN       = 2,
    MGL_WIN_DARKYELLOW      = 3,
    MGL_WIN_DARKBLUE        = 4,
    MGL_WIN_DARKMAGENTA     = 5,
    MGL_WIN_DARKCYAN        = 6,
    MGL_WIN_LIGHTGRAY       = 7,
    MGL_WIN_TURQUOISE        = 8,
    MGL_WIN_SKYBLUE         = 9,
    MGL_WIN_CREAM           = 246,
    MGL_WIN_MEDIUMGRAY      = 247,
    MGL_WIN_DARKGRAY        = 248,
    MGL_WIN_LIGHTRED        = 249,
    MGL_WIN_LIGHTGREEN      = 250,
    MGL_WIN_LIGHTYELLOW     = 251,
    MGL_WIN_LIGHTBLUE       = 252,
    MGL_WIN_LIGHTMAGENTA    = 253,
    MGL_WIN_LIGHTCYAN       = 254,
    MGL_WIN_WHITE           = 255
} MGL_WIN_COLORS
```

Prototype In mgraph.h

Description

Defines the Windows standard colors for 256 color graphics modes when in a window. 8,9,246,247 are reserved and you should not count on these colors always being the same. For 16 color bitmaps, colors 248-255 are mapped to colors 8-15.

MGL_backModes

Declaration

```
typedef enum {
    MGL_OPAQUE_BACKGROUND,
    MGL_TRANSPARENT_BACKGROUND
} MGL_backModes
```

Prototype In

mgraph.h

Description

Defines the background mode for monochrome bitmap and font drawing. By default the MGL starts up in transparent mode, however you can change the background mode to draw monochrome bitmaps with all 0 bits draw in the background color instead of being transparent. This affects all monochrome bitmap functions, including text rendering.

Members

<i>MGL_OPAQUE_BACKGROUND</i>	Background for monochrome bitmaps is opaque, and drawn in the current background color.
<i>MGL_TRANSPARENT_BACKGROUND</i>	Background for monochrome bitmaps is transparent

MGL_bitBltFxFlagsType

Declaration

```
typedef enum {
    MGL_BLT_WRITE_MODE_ENABLE           = 0x00000001,
    MGL_BLT_STRETCH_NEAREST             = 0x00000002,
    MGL_BLT_STRETCH_XINTERP             = 0x00000004,
    MGL_BLT_STRETCH_YINTERP             = 0x00000008,
    MGL_BLT_COLOR_KEY_SRC_SINGLE        = 0x00000010,
    MGL_BLT_COLOR_KEY_SRC_RANGE         = 0x00000020,
    MGL_BLT_COLOR_KEY_DST_SINGLE        = 0x00000040,
    MGL_BLT_COLOR_KEY_DST_RANGE         = 0x00000080,
    MGL_BLT_FLIPX                       = 0x00000100,
    MGL_BLT_FLIPY                       = 0x00000200,
    MGL_BLT_BLEND                       = 0x00000400,
    MGL_BLT_DITHER                      = 0x00002000,
    MGL_BLT_ANY_STRETCH                 = MGL_BLT_STRETCH_NEAREST
    | MGL_BLT_STRETCH_XINTERP | MGL_BLT_STRETCH_YINTERP,
    MGL_BLT_ANY_SINGLE_COLOR_KEY        =
MGL_BLT_COLOR_KEY_SRC_SINGLE | MGL_BLT_COLOR_KEY_DST_SINGLE,
    MGL_BLT_ANY_RANGE_COLOR_KEY         =
MGL_BLT_COLOR_KEY_SRC_RANGE | MGL_BLT_COLOR_KEY_DST_RANGE,
    MGL_BLT_ANY_COLOR_KEY               =
MGL_BLT_ANY_SINGLE_COLOR_KEY | MGL_BLT_ANY_RANGE_COLOR_KEY
} MGL_bitBltFxFlagsType
```

Prototype In mgraph.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.

The MGL_BLT_WRITE_MODE_ENABLE flag enables logical write modes for extended BitBlt functions.

The MGL_BLT_STRETCH_NEAREST flag enables stretching with nearest pixel filtering.

The MGL_BLT_STRETCH_XINTERP flag enables stretching with linearly interpolated filtering in the X direction.

The MGL_BLT_STRETCH_YINTERP flag enables stretching with linearly interpolated filtering in the Y direction.

The MGL_BLT_COLOR_KEY_SRC_SINGLE flag enables source transparent color keying with single color key. When source color keying is enabled, any pixel data in the incoming bitmap that matches the color key value in colorKeyLo value will be ignored

and not drawn to the destination surface. This essentially makes those source pixels transparent.

The MGL_BLT_COLOR_KEY_SRC_RANGE flag enables source transparent color keying with a range of color keys values. This is the same as single source color keying, but the color key values may be allows to fall within a range of available colors defined in colorKeyLo and colorKeyHi. This is useful if the data has been filtered causing the colors to shift slightly.

The MGL_BLT_COLOR_KEY_DST_SINGLE flag enables destination transparent color keying with single color key. When destination color keying is enabled (sometimes called blue-screening), any destination pixels in the framebuffer that match the color key value in colorKeyLo, will cause the source input pixels to be ignored.

The MGL_BLT_COLOR_KEY_DST_RANGE flag enables destination transparent color keying with a range of color keys values. This is the same as single destination color keying, but the color key values may be allows to fall within a range of available colors defined in colorKeyLo and colorKeyHi.

The MGL_BLT_FLIPX flag enables bitmap flipping 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 MGL_BLT_FLIPY flag enables bitmap flipping 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 MGL_BLT_BLEND flag enables alpha blending. When you enable alpha blending the values in srcBlendfunc and dstBlendFunc members of *bltfx_t* are used to determine the blending operation to apply to the pixels.

The MGL_BLT_DITHER flag enables dithering when color converting an RGB bitmap to a color depth of a lower pixel depth. This will occur when converting any RGB bitmap (ie: 15, 16, 24 or 32-bit) to a color index pixel format, or when converting 24 or 32-bit bitmaps to 15 or 16-bit bitmaps. If dithering is not enabled, the closest color to the source pixel will be found and drawn into the framebuffer. Dithering slows things down somewhat when dithering to 15/16-bit destination bitmaps, but produces better quality. Dithering down to 8-bit bitmaps looks best if a halftone palette is used, and in fact is a lot faster than using the closest color method (which has to search the color palette for every pixel drawn). Dithering to 8-bit bitmaps will however map to any palette, but the quality is best if a halftone palette is used.

Note: *For the most part any feature can be combined with any other feature with the MGL_bitBltFx family of functions. However some features are mutually exclusive, such as blending a logical write modes. Also blending is only available if the destination pixel format is not a color index pixel format.*

Members

MGL_BLT_WRITE_MODE_ENABLE	Write mode enabled
MGL_BLT_STRETCH_NEAREST	Enable stretching, nearest pixel
MGL_BLT_STRETCH_XINTERP	Enable X axis filtering for stretch blit
MGL_BLT_STRETCH_YINTERP	Enable Y axis filtering for stretch blit
MGL_BLT_COLOR_KEY_SRC_SINGL E	Source color keying enabled, single color
MGL_BLT_COLOR_KEY_SRC_RANG E	Source color keying enabled, range of colors
MGL_BLT_COLOR_KEY_DST_SINGL E	Destination color keying enabled, single color
MGL_BLT_COLOR_KEY_DST_RANG E	Destination color keying enabled, range of colors
MGL_BLT_FLIPX	Enable flip in X axis
MGL_BLT_FLIPY	Enable flip in Y axis
MGL_BLT_BLEND	Enable alpha blending
MGL_BLT_DITHER	Dither if an 8/15/16bpp destination
MGL_BLT_ANY_STRETCH	Flags that any stretching is enabled
MGL_BLT_ANY_SINGLE_COLOR_KEY Y	Flags that any single color key is enabled
MGL_BLT_ANY_RANGE_COLOR_KEY Y	Flags that any range color key is enabled
MGL_BLT_ANY_COLOR_KEY	Flags that any color key is enabled

MGL_blendFuncType

Declaration

```
typedef enum {
    MGL_BLEND_NONE,
    MGL_BLEND_ZERO,
    MGL_BLEND_ONE,
    MGL_BLEND_SRCOLOR,
    MGL_BLEND_ONEMINUSSRCOLOR,
    MGL_BLEND_SRCALPHA,
    MGL_BLEND_ONEMINUSSRCALPHA,
    MGL_BLEND_DSTALPHA,
    MGL_BLEND_ONEMINUSDSTALPHA,
    MGL_BLEND_DSTCOLOR,
    MGL_BLEND_ONEMINUSDSTCOLOR,
    MGL_BLEND_SRCALPHASATURATE,
    MGL_BLEND_CONSTANTCOLOR,
    MGL_BLEND_ONEMINUSCONSTANTCOLOR,
    MGL_BLEND_CONSTANTALPHA,
    MGL_BLEND_ONEMINUSCONSTANTALPHA,
    MGL_BLEND_SRCALPHAFAST,
    MGL_BLEND_CONSTANTALPHAFAST
} MGL_blendFuncType
```

Prototype In mgraph.h

Description

Flags for 2D alpha blending functions supported by the SciTech MGL. The values in here define the the alpha blending functions passed to the *MGL_setBlendFunc* function. Essentially the blend function defines how to combine the source and destination pixel colors 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 *MGL_BLEND_CONSTANTALPHA*, the *SrcFunc* above becomes:

$$\text{SrcFunc} = \text{ConstAlpha}$$

If the destination alpha blending function is set to *MGL_BLEND_ONEMINUSDSTALPHA* 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 is usually used:

$$\text{DstColor} = \text{SrcColor} * \text{SrcAlpha} + \text{DstColor} * (1 - \text{SrcAlpha})$$

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 MGL_BLEND_SRCALPHAFAST 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 is usually used:

$$\text{DstColor} = \text{SrcColor} * \text{ConstantAlpha} + \text{DstColor} * (1 - \text{ConstantAlpha})$$

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 MGL_BLEND_CONSTANTALPHAFAST 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 MGL_setColor, and the constant alpha value set by a call to MGL_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 write mode. Logical write modes and blending cannot be used at the same time.*

Members

MGL_BLEND_NONE

No alpha blending

MGL_BLEND_ZERO

Blend factor is always zero

MGL_BLEND_ONE	Blend factor is always one
MGL_BLEND_SRCCOLOR	Blend factor is source color
MGL_BLEND_ONEMINUSSRCCOLOR	Blend factor is 1-source color
MGL_BLEND_SRCALPHA	Blend factor is source alpha
MGL_BLEND_ONEMINUSSRCALPHA	Blend factor is 1-source alpha
MGL_BLEND_DSTALPHA	Blend factor is destination alpha
MGL_BLEND_ONEMINUSDSTALPHA	Blend factor is 1-destination alpha
MGL_BLEND_DSTCOLOR	Blend factor is destination color
MGL_BLEND_ONEMINUSDSTCOLOR	Blend factor is 1-destination color
MGL_BLEND_SRCALPHASATURATE	Blend factor is src alpha saturation
MGL_BLEND_CONSTANTCOLOR	Blend factor is a constant color
MGL_BLEND_ONEMINUSCONSTANTCOLOR	Blend factor is 1-constant color
MGL_BLEND_CONSTANTALPHA	Blend factor is constant alpha
MGL_BLEND_ONEMINUSCONSTANTALPHA	Blend factor is 1-constant alpha
MGL_BLEND_SRCALPHAFAST	Common case of optimised src alpha
MGL_BLEND_CONSTANTALPHAFAST	Common case of optimised constant alpha

MGL_bufferFlagsType

Declaration

```
typedef enum {
    MGL_BUF_SYSMEM           = 0x00000001,
    MGL_BUF_CACHED           = 0x00000002,
    MGL_BUF_MOVEABLE         = 0x00000004,
    MGL_BUF_PAGEABLE         = 0x00000008,
    MGL_BUF_PRIORITY         = 0x00000010,
    MGL_BUF_NOSYSMEM         = 0x00000020
} MGL_bufferFlagsType
```

Prototype In

mgraph.h

Description

Defines the flags passed to *MGL_createBuffer*. The flags define how the buffer is allocated by the MGL, and the type of buffer.

The MGL_BUF_SYSMEM flag indicates that the buffer is currently located in system memory only. It is possible for a buffer that was allocated with the MGL_BUF_PAGEABLE and MGL_BUF_CACHED 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 MGL_BUF_CACHED 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 MGL, 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 *MGL_updateBufferCache* and *MGL_updateFromBufferCache* functions to keep the system memory cache in sync as necessary. Note also that some of the buffer manager drawing functions will draw from the buffer cache if it is present and the hardware cannot accelerate the operation within video memory. Hence it is important that the application keep the buffer cache coherent with the contents of the video memory buffer or some drawing operations may come out incorrect.

The MGL_BUF_MOVEABLE 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 MGL_BUF_PAGEABLE 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 MGL_BUF_PAGEABLE flag will automatically set the MGL_BUF_CACHED 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 freed.

The MGL_BUF_PRIORITY 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.

The MGL_BUF_NOSYSMEM 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 MGL_BUF_CACHED or MGL_BUF_PAGEABLE flags are set).

Members

<i>MGL_BUF_SYSMEM</i>	Buffer is currently located in system memory
<i>MGL_BUF_CACHED</i>	Buffer is cached in system memory
<i>MGL_BUF_MOVEABLE</i>	Buffer can be moved around to compact buffer heap
<i>MGL_BUF_PAGEABLE</i>	Buffer can be paged to system memory
<i>MGL_BUF_PRIORITY</i>	Buffer is a high priority bitmap
<i>MGL_BUF_NOSYSMEM</i>	Buffer should never be allocated in system memory

MGL_ditherModes

Declaration

```
typedef enum {  
    MGL_DITHER_OFF,  
    MGL_DITHER_ON  
} MGL_ditherModes
```

Prototype In

mgraph.h

Description

Defines the MGL dithering modes, used when blitting RGB images to 8, 15 and 16 bits per pixel device context. If dithering is enabled, the blit operation will dither the resulting image to produce the best quality. When dithering is disabled, the blit operation uses the closest color which has less quality but is faster.

Note: *The closest color method is fastest when the destination device context is a 15 or 16 bits per pixel bitmap. However when the destination is an 8 bits per pixel device context, the dithering mode will usually be faster.*

Note: *Dithering is on by default in the MGL.*

MGL_errorType

Declaration

```
typedef enum {
    grOK                = 0,
    grNoInit            = -1,
    grNotDetected       = -2,
    grDriverNotFound    = -3,
    grBadDriver         = -4,
    grLoadMem           = -5,
    grInvalidMode       = -6,
    grError             = -8,
    grInvalidName       = -9,
    grNoMem             = -10,
    grNoModeSupport     = -11,
    grInvalidFont       = -12,
    grBadFontFile       = -13,
    grFontNotFound      = -14,
    grOldDriver         = -15,
    grInvalidDevice     = -16,
    grInvalidDC         = -17,
    grInvalidCursor     = -18,
    grCursorNotFound    = -19,
    grInvalidIcon       = -20,
    grIconNotFound      = -21,
    grInvalidBitmap     = -22,
    grBitmapNotFound    = -23,
    grNewFontFile       = -25,
    grNoDoubleBuff      = -26,
    grNoHardwareBlt     = -28,
    grNoOffscreenMem    = -29,
    grInvalidPF         = -30,
    grInvalidBuffer     = -31,
    grNoDisplayDC       = -32,
    grFailLoadRef2d     = -33,
    grLastError         = -34
} MGL_errorType
```

Prototype In

mgraph.h

Description

Defines the error codes returned by *MGL_result*. If a function fails for any reason, you can check the error code return by *MGL_result* to determine the cause of the failure, or display an error message to the user with *MGL_errorMsg*.

Members

<i>grOK</i>	No error
<i>grNoInit</i>	Graphics driver has not been installed
<i>grNotDetected</i>	Graphics hardware was not detected

<i>grDriverNotFound</i>	Graphics driver file not found
<i>grBadDriver</i>	File loaded was not a graphics driver
<i>grLoadMem</i>	Not enough memory to load graphics driver
<i>grInvalidMode</i>	Invalid graphics mode for selected driver
<i>grError</i>	General graphics error
<i>grInvalidName</i>	Invalid driver name
<i>grNoMem</i>	Not enough memory to perform operation
<i>grNoModeSupport</i>	Select video mode not supported by hard.
<i>grInvalidFont</i>	Invalid font data
<i>grBadFontFile</i>	File loaded was not a font file
<i>grFontNotFound</i>	Font file was not found
<i>grOldDriver</i>	Driver file is an old version
<i>grInvalidDevice</i>	Invalid device for selected operation
<i>grInvalidDC</i>	Invalid device context
<i>grInvalidCursor</i>	Invalid cursor file
<i>grCursorNotFound</i>	Cursor file was not found
<i>grInvalidIcon</i>	Invalid icon file
<i>grIconNotFound</i>	Icon file was not found
<i>grInvalidBitmap</i>	Invalid bitmap file
<i>grBitmapNotFound</i>	Bitmap file was not found
<i>grNewFontFile</i>	Only Windows 2.x font files supported
<i>grNoDoubleBuff</i>	Double buffering is not available
<i>grNoHardwareBlt</i>	No hardware bitBlt for OffscreenDC
<i>grNoOffscreenMem</i>	No available offscreen memory
<i>grInvalidPF</i>	Invalid pixel format for memory DC
<i>grInvalidBuffer</i>	Invalid offscreen buffer
<i>grNoDisplayDC</i>	Display DC has not been created
<i>grFailLoadRef2d</i>	2D reference rasteriser driver failed to load
<i>grLastError</i>	Last error number in list

MGL_fontBlendType

Declaration

```
typedef enum {  
    MGL_AA_NORMAL,  
    MGL_AA_RGBBLEND  
} MGL_fontBlendType
```

Prototype In

mgraph.h

Description

Determines the type of blending used when drawing anti-aliased fonts. Currently only two types, which are blended and non Blended. Used by the *MGL_setFontBlendMode* function, and only affects TrueType or Adobe Type 1 fonts loaded with anti-aliasing enabled.

Members

- | | |
|------------------------|---|
| <i>MGL_AA_NORMAL</i> | Four level antialiasing, between foreground and background. |
| <i>MGL_AA_RGBBLEND</i> | Blends antialiasing with contents of screen. RGB only. |

MGL_fontLibType

Declaration

```
typedef enum {  
    MGL_BITMAPFONT_LIB,  
    MGL_TRUETYPEFONT_LIB,  
    MGL_TYPE1FONT_LIB  
} MGL_fontLibType
```

Prototype In

mgraph.h

Description

Defines the different font library types

Members

<i>MGL_BITMAPFONT_LIB</i>	Bitmap font library (Windows 2.x style)
<i>MGL_TRUETYPEFONT_LIB</i>	TrueType scalable font library
<i>MGL_TYPE1FONT_LIB</i>	Adobe Type 1 scalable font library

MGL_fontType

Declaration

```
typedef enum {
    MGL_VECTORFONT          = 1,
    MGL_FIXEDFONT,
    MGL_PROPFONT,
    MGL_ANTIALIASEDFONT = 0x80,
    MGL_FONTTYPEMASK      = 0x3F
} MGL_fontType
```

Prototype In

mgraph.h

Description

Defines the different font types

Members

<i>MGL_VECTORFONT</i>	Vector font
<i>MGL_FIXEDFONT</i>	Fixed width bitmap font
<i>MGL_PROPFONT</i>	Proportional width bitmap font
<i>MGL_ANTIALIASEDFONT</i>	Flags that a font is anti-aliased
<i>MGL_FONTTYPEMASK</i>	Mask for the font type flags

MGL_glContextFlagsType

Declaration

```
typedef enum {
    MGL_GL_VISUAL                = 0x8000,
    MGL_GL_FORCEMEM              = 0x4000,
    MGL_GL_RGB                   = 0x0000,
    MGL_GL_INDEX                 = 0x0001,
    MGL_GL_SINGLE                = 0x0000,
    MGL_GL_DOUBLE                = 0x0002,
    MGL_GL_ACCUM                 = 0x0004,
    MGL_GL_ALPHA                 = 0x0008,
    MGL_GL_DEPTH                 = 0x0010,
    MGL_GL_STENCIL               = 0x0020,
    MGL_GL_STEREO                = 0x0040
} MGL_glContextFlagsType
```

Prototype In

mgraph.h

Description

MGL_glCreateContext flags to initialize the pixel format used by the OpenGL rendering context. If you pass in *MGL_GL_VISUAL*, the visual used will be the one currently selected by the previous call to *MGL_glSetVisual*, and provides the application programmer with complete control over the pixel formats used.

You can pass in a combination of any of the other flags (ie: *MGL_GL_RGB* | *MGL_GL_DOUBLE* | *MGL_GL_DEPTH*) to let the MGL know what you want and to have it automatically select an appropriate visual for you. This provides a quick and simple way to get application code up and running.

Members

<i>MGL_GL_VISUAL</i>	Use currently assigned visual from call to <i>MGL_glSetVisual</i>
<i>MGL_GL_FORCEMEM</i>	Force system memory back buffer for all rendering
<i>MGL_GL_RGB</i>	Select RGB rendering mode (/ default/)
<i>MGL_GL_INDEX</i>	Select color index display mode
<i>MGL_GL_SINGLE</i>	Select single buffered display mode (/ default/)
<i>MGL_GL_DOUBLE</i>	Select double buffered display mode
<i>MGL_GL_ACCUM</i>	Enable accumulation buffer (16 bits)
<i>MGL_GL_ALPHA</i>	Enable alpha buffer (8 bit)
<i>MGL_GL_DEPTH</i>	Enable depth buffer (16 bits)
<i>MGL_GL_STENCIL</i>	Enable stencil buffer (8 bits)
<i>MGL_GL_STEREO</i>	Enable stereo mode

MGL_glOpenGLType

Declaration

```
typedef enum {
    MGL_GL_AUTO,
    MGL_GL_DEFAULT,
    MGL_GL_SGI,
    MGL_GL_MESA,
    MGL_GL_GLDIRECT,
    MGL_GL_GLDIRECTCAD
} MGL_glOpenGLType
```

Prototype In

mgraph.h

Description

MGL_glSetOpenGL flags to select the OpenGL implementation. In the AUTO mode we automatically determine which version of OpenGL to use depending on the target runtime system. Unless there is hardware acceleration available via the OS default OpenGL, we choose Silicon Graphics OpenGL. If the Silicon Graphics OpenGL is not available, we fall back on the Mesa software rendering libraries.

Members

<i>MGL_GL_AUTO</i>	Automatically choose OpenGL implementation
<i>MGL_GL_DEFAULT</i>	Force OS default OpenGL implementation (usually hardware)
<i>MGL_GL_SGI</i>	Force SGI software OpenGL implementation
<i>MGL_GL_MESA</i>	Force Mesa software OpenGL implementation
<i>MGL_GL_GLDIRECT</i>	Force SciTech GLDirect OpenGL game driver implementation
<i>MGL_GL_GLDIRECTCAD</i>	Force SciTech GLDirect OpenGL CAD driver implementation

MGL_hardwareFlagsType

Declaration

```
typedef enum {
    MGL_HW_NONE                = 0x00000000,
    MGL_HW_LINE                 = 0x00000010,
    MGL_HW_STIPPLE_LINE        = 0x00000020,
    MGL_HW_POLY                 = 0x00000040,
    MGL_HW_RECT                 = 0x00000080,
    MGL_HW_PATT_RECT            = 0x00000100,
    MGL_HW_CLRPATT_RECT         = 0x00000200,
    MGL_HW_SYS_BLT              = 0x00000400,
    MGL_HW_SCR_BLT              = 0x00000800,
    MGL_HW_SRCTRANS_BLT         = 0x00001000,
    MGL_HW_DSTTRANS_BLT         = 0x00002000,
    MGL_HW_SRCTRANS_SYS_BLT     = 0x00004000,
    MGL_HW_DSTTRANS_SYS_BLT     = 0x00008000,
    MGL_HW_STRETCH_BLT          = 0x00010000,
    MGL_HW_STRETCH_SYS_BLT      = 0x00020000,
    MGL_HW_MONO_BLT             = 0x00040000,
    MGL_HW_FLAGS                 = 0x0007FFF0
} MGL_hardwareFlagsType
```

Prototype In

mgraph.h

Description

Defines the flags for the types of hardware acceleration supported by the device context. This will allow the application to tailor the use of MGL functions depending upon whether specific hardware support is available. Hence applications can use specialised software rendering support if the desired hardware support is not available on the end user system.

Note: *If the hardware flags are not MGL_HW_NONE, you must call the MGL_beginDirectAccess and MGL_endDirectAccess functions before and after any custom code that does direct framebuffer rendering!! This is not necessary for non-accelerated device contexts, so you might want to optimise these calls out if there is no hardware acceleration support.*

Members

MGL_HW_NONE	No hardware acceleration
MGL_HW_LINE	Hardware line drawing
MGL_HW_STIPPLE_LINE	Hardware stippled line drawing
MGL_HW_POLY	Hardware polygon filling
MGL_HW_RECT	Hardware rectangle fill
MGL_HW_PATT_RECT	Hardware pattern rectangle fill
MGL_HW_CLRPATT_RECT	Hardware color pattern fill
MGL_HW_SYS_BLT	Hardware system->screen bitBlt
MGL_HW_SCR_BLT	Hardware screen->screen bitBlt

<i>MGL_HW_SRCTRANS_BLT</i>	Hardware source transparent blt
<i>MGL_HW_DSTTRANS_BLT</i>	Hardware dest. transparent blt
<i>MGL_HW_SRCTRANS_SYS_BLT</i>	Hardware system->screen source transparent blt
<i>MGL_HW_DSTTRANS_SYS_BLT</i>	Hardware system->screen destination transparent blt
<i>MGL_HW_STRETCH_BLT</i>	Hardware stretch blt
<i>MGL_HW_STRETCH_SYS_BLT</i>	Hardware system->screen stretch blt
<i>MGL_HW_MONO_BLT</i>	Hardware monochrome blt

MGL_lineStyleType

Declaration

```
typedef enum {
    MGL_LINE_PENSTYLE,
    MGL_LINE_STIPPLE
} MGL_lineStyleType
```

Prototype In

mgraph.h

Description

Defines the line styles passed to *MGL_setLineStyle*.

Members

<i>MGL_LINE_PENSTYLE</i>	Line drawn in current pen style
<i>MGL_LINE_STIPPLE</i>	Line drawn with current line stipple pattern. The line stipple pattern is a 16x1 pattern that defines which pixels should be drawn in the line. Where a bit is a 1 in the pattern, a pixel will be drawn and where a bit is a 0 in the pattern, the pixel will be left untouched on the screen.

MGL_modeFlagsType

Declaration

```
typedef enum {
    MGL_HAVE_LINEAR           = 0x00000001,
    MGL_HAVE_REFRESH_CTRL    = 0x00000002,
    MGL_HAVE_INTERLACED      = 0x00000004,
    MGL_HAVE_DOUBLE_SCAN     = 0x00000008,
    MGL_HAVE_TRIPLEBUFFER    = 0x00000010,
    MGL_HAVE_STEREO          = 0x00000020,
    MGL_HAVE_STEREO_DUAL     = 0x00000040,
    MGL_HAVE_STEREO_HWSYNC   = 0x00000080,
    MGL_HAVE_STEREO_EVCSYNC  = 0x00000100,
    MGL_HAVE_HWCURSOR        = 0x00000200,
    MGL_HAVE_ACCEL_2D        = 0x00000400,
    MGL_HAVE_ACCEL_3D        = 0x00000800,
    MGL_HAVE_ACCEL_VIDEO     = 0x00001000,
    MGL_HAVE_VIDEO_XINTERP   = 0x00002000,
    MGL_HAVE_VIDEO_YINTERP   = 0x00004000,
    MGL_IS_COLOR_INDEX       = 0x00008000
} MGL_modeFlagsType
```

Prototype In

mgraph.h

Description

Defines the flags returned by the *MGL_modeFlags* functions. This function allows you to enumerate and detect support for different types of hardware features for a specific graphics mode after calling *MGL_detectGraph*, but before you actually initialize the desired mode. This will allow your application to search for fullscreen graphics modes that have the features that you desire (such as 2D or 3D acceleration).

Members

<i>MGL_HAVE_LINEAR</i>	Graphics mode supports a hardware linear framebuffer.
<i>MGL_HAVE_REFRESH_CTRL</i>	Graphics mode supports refresh rate control, allowing you to increase the refresh rate to a desired value (such as high refresh rates for stereo LC shutter glasses support).
<i>MGL_HAVE_INTERLACED</i>	Graphics mode supports interlaced operation, and you can request and interlaced mode via the refresh rate control mechanism in the MGL.
<i>MGL_HAVE_DOUBLE_SCAN</i>	Graphics mode supports double scan operation.
<i>MGL_HAVE_TRIPLEBUFFER</i>	Graphics mode supports hardware triple buffering, allowing your application to use true triple buffering without any visible flickering.

<i>MGL_HAVE_STEREO</i>	Graphics mode supports hardware stereo page flipping, providing hardware support for stereo LC shutter glasses.
<i>MGL_HAVE_STEREO_DUAL</i>	Graphics mode supports hardware stereo page flipping, with dual display start addresses.
<i>MGL_HAVE_STEREO_HWSYNC</i>	Graphics mode provides hardware stereo sync support via an external connector for stereo LC shutter glasses.
<i>MGL_HAVE_STEREO_EVCSYNC</i>	Graphics mode provides support for the EVC stereo connector. If this bit is set, the above bit will also be set.
<i>MGL_HAVE_HWCURSOR</i>	Graphics mode supports a hardware cursor.
<i>MGL_HAVE_ACCEL_2D</i>	Graphics mode supports 2D hardware acceleration. 2D acceleration may be provided either by WinDirect and a VESA VBE/AF driver, or via DirectDraw.
<i>MGL_HAVE_ACCEL_3D</i>	Graphics mode supports 3D hardware acceleration. Hardware 3D acceleration is always provided in the form of an OpenGL hardware driver of some form.
<i>MGL_HAVE_ACCEL_VIDEO</i>	Graphics mode supports hardware video acceleration, either via WinDirect and a VESA VBE/AF driver, or via DirectDraw.
<i>MGL_HAVE_VIDEO_XINTERP</i>	Graphics mode supports hardware video with interpolation along the X axis.
<i>MGL_HAVE_VIDEO_YINTERP</i>	Graphics mode supports hardware video with interpolation along the Y axis.
<i>MGL_IS_COLOR_INDEX</i>	Indicates that the mode is a color index mode

MGL_palRotateType

Declaration

```
typedef enum {  
    MGL_ROTATE_UP,  
    MGL_ROTATE_DOWN  
} MGL_palRotateType
```

Prototype In

mgraph.h

Description

Defines the different palette rotation directions

Members

<i>MGL_ROTATE_UP</i>	Rotate the palette values up
<i>MGL_ROTATE_DOWN</i>	Rotate the palette values down

MGL_penStyleType

Declaration

```
typedef enum {
    MGL_BITMAP_SOLID,
    MGL_BITMAP_OPAQUE,
    MGL_BITMAP_TRANSPARENT,
    MGL_PIXMAP,
    MGL_PIXMAP_TRANSPARENT
} MGL_penStyleType
```

Prototype In

mgraph.h

Description

Defines the pen styles passed to *MGL_setPenStyle*. These styles define the different fill styles that can be used when the filling the interior of filled primitives and also the outline of non-filled primitives.

Members

<i>MGL_BITMAP_SOLID</i>	Fill with a solid color
<i>MGL_BITMAP_OPAQUE</i>	Fill with an opaque bitmap pattern. Where bits in the pattern are a 1, the foreground color is used. Where bits in the pattern are a 0, the background color is used. The pattern itself is defined as an 8x8 monochrome bitmap.
<i>MGL_BITMAP_TRANSPARENT</i>	Fill with a transparent bitmap pattern. Where bits in the pattern are a 1, the foreground color is used. Where bits in the pattern are a 0, the pixel is left unmodified on the screen. The pattern itself is defined as an 8x8 monochrome bitmap.
<i>MGL_PIXMAP</i>	Fill with a color pixmap pattern. The pixmap pattern is defined as an 8x8 array of <i>color_t</i> values, where each entry corresponds to the color values packed for the appropriate color mode (ie: a color index in color index modes and a packed RGB value in HiColor and TrueColor modes).
<i>MGL_PIXMAP_TRANSPARENT</i>	Fill with a color pixmap pattern, but with one of the colors to be transparent. The transparent color is set with a call to <i>MGL_setPenPixmapTransparent</i> .

MGL_polygonType

Declaration

```
typedef enum {
    MGL_CONVEX_POLYGON,
    MGL_COMPLEX_POLYGON,
    MGL_AUTO_POLYGON
} MGL_polygonType
```

Prototype In

mgraph.h

Description

Defines the polygon types passed to *MGL_setPolygonType*.

Members

- | | |
|-----------------------------------|---|
| <i>MGL_CONVEX_POLYGON</i> | Monotone vertical polygon (includes convex polygons). A monotone vertical polygon is one whereby there will never be a horizontal line that can intersect the polygon at more than two edges at a time. Note that if you set the polygon type to this value and you pass it a non-monotone vertical polygon, the output results are undefined. |
| <i>MGL_COMPLEX_POLYGON</i> | Non-Simple polygons. When set to this mode the MGL will correctly rasterise all polygon types that you pass to it, however the drawing will be slower. |
| <i>MGL_AUTO_POLYGON</i> | Auto detect the polygon type. In this mode the MGL will examine the polygon vertices passed in and will automatically draw it with the faster routines if it is monotone vertical. Note that this does incur an overhead for the checking code, so if you know all your polygons are monotone vertical or convex, then you should set the type to <i>MGL_CONVEX_POLYGON</i> . |

MGL_refreshRateType

Declaration

```
typedef enum {  
    MGL_DEFAULT_REFRESH = -1,  
    MGL_INTERLACED_MODE = 0x4000  
} MGL_refreshRateType
```

Prototype In

mgraph.h

Description

Defines the flags passed to MGL_setRefreshRate. You can pass the value of MGL_DEFAULT_REFRESH to set the refresh rate to the adapter default.

Members

MGL_DEFAULT_REFRESH Use the default refresh rate for the graphics mode
MGL_INTERLACED_MODE Set the mode to be interlaced (not always supported)

MGL_rop3CodesType

Declaration

```
typedef enum {
    MGL_R3_0,
    MGL_R3_DPSoon,
    MGL_R3_DPSona,
    MGL_R3_PSon,
    MGL_R3_SDPona,
    MGL_R3_DPon,
    MGL_R3_PDSxnon,
    MGL_R3_PDSaon,
    MGL_R3_SDPnaa,
    MGL_R3_PDSxon,
    MGL_R3_DPna,
    MGL_R3_PSDnaon,
    MGL_R3_SPna,
    MGL_R3_PDSnaon,
    MGL_R3_PDSonon,
    MGL_R3_Pn,
    MGL_R3_PDSona,
    MGL_R3_DSon,
    MGL_R3_SDPxnon,
    MGL_R3_SDPaon,
    MGL_R3_DPSxnon,
    MGL_R3_DPSaon,
    MGL_R3_PSDPSanaxx,
    MGL_R3_SSPxDSxaxn,
    MGL_R3_SPxPDxa,
    MGL_R3_SDPSanaxn,
    MGL_R3_PDSPaox,
    MGL_R3_SDPsaxn,
    MGL_R3_PSDPaox,
    MGL_R3_DSPDxaxn,
    MGL_R3_PDSox,
    MGL_R3_PDSoan,
    MGL_R3_DPSnaa,
    MGL_R3_SDPxon,
    MGL_R3_DSna,
    MGL_R3_SPDnaon,
    MGL_R3_SPxDSxa,
    MGL_R3_PDSPanaxn,
    MGL_R3_SDPsaox,
    MGL_R3_SDPsxn timer,
    MGL_R3_DPSxa,
    MGL_R3_PSDPSaox,
    MGL_R3_DPSana,
    MGL_R3_SSPxPDxaxn,
    MGL_R3_SPDSaox,
    MGL_R3_PSDnox,
    MGL_R3_PSDPxox,
```

MGL_R3_PSDnoan,
 MGL_R3_PSna,
 MGL_R3_SDPnaon,
 MGL_R3_SDPSoox,
 MGL_R3_Sn,
 MGL_R3_SPDSaox,
 MGL_R3_SPDSxnox,
 MGL_R3_SDPox,
 MGL_R3_SDPoan,
 MGL_R3_PSDPoax,
 MGL_R3_SPDnox,
 MGL_R3_SPDSxox,
 MGL_R3_SPDnoan,
 MGL_R3_Psx,
 MGL_R3_SPDSonox,
 MGL_R3_SPDSnaox,
 MGL_R3_PSan,
 MGL_R3_PSDnaa,
 MGL_R3_DPSxon,
 MGL_R3_SDxPDxa,
 MGL_R3_SPDSanaxn,
 MGL_R3_SDna,
 MGL_R3_DPSnaon,
 MGL_R3_DSPDaiox,
 MGL_R3_PSDPxaxn,
 MGL_R3_SDPxa,
 MGL_R3_PDSPDaooxn,
 MGL_R3_DPSDoax,
 MGL_R3_PDSnox,
 MGL_R3_SDPana,
 MGL_R3_SSPxDSxoxn,
 MGL_R3_PDSPxox,
 MGL_R3_PDSnoan,
 MGL_R3_PDna,
 MGL_R3_DSPnaon,
 MGL_R3_DSPDaiox,
 MGL_R3_SPDSxaxn,
 MGL_R3_DPSonon,
 MGL_R3_Dn,
 MGL_R3_DPSox,
 MGL_R3_DPSoan,
 MGL_R3_PDSPoax,
 MGL_R3_DPSnox,
 MGL_R3_DPx,
 MGL_R3_DSPDonox,
 MGL_R3_DSPDxox,
 MGL_R3_DPSnoan,
 MGL_R3_DSPDnaox,
 MGL_R3_DPan,
 MGL_R3_PDSxa,
 MGL_R3_DSPDSaooxn,
 MGL_R3_DSPDoax,

MGL_R3_SDPnox,
 MGL_R3_SDPSoax,
 MGL_R3_DSPnox,
 MGL_R3_DSx,
 MGL_R3_SDPSonox,
 MGL_R3_DSPDSonoxxn,
 MGL_R3_PDSxxn,
 MGL_R3_DPSax,
 MGL_R3_PSDPSoaxxn,
 MGL_R3_SDPax,
 MGL_R3_PDSPDoaxxn,
 MGL_R3_SDPSnoax,
 MGL_R3_PDSxnan,
 MGL_R3_PDSana,
 MGL_R3_SSDxPDxaxn,
 MGL_R3_SDPSxox,
 MGL_R3_SDPnoan,
 MGL_R3_DSPDxox,
 MGL_R3_DSPnoan,
 MGL_R3_SDPSnaox,
 MGL_R3_DSan,
 MGL_R3_PDSax,
 MGL_R3_DSPDSoaxxn,
 MGL_R3_DPSDnoax,
 MGL_R3_SDPxnan,
 MGL_R3_SPDSnoax,
 MGL_R3_DPSxnan,
 MGL_R3_SPxDSxo,
 MGL_R3_DPSaan,
 MGL_R3_DPSaa,
 MGL_R3_SPxDSxon,
 MGL_R3_DPSxna,
 MGL_R3_SPDSnoaxn,
 MGL_R3_SDPxna,
 MGL_R3_PDSPnoaxn,
 MGL_R3_DSPDSoaxx,
 MGL_R3_PDSaxn,
 MGL_R3_DSa,
 MGL_R3_SDPSnaoxn,
 MGL_R3_DSPnoa,
 MGL_R3_DSPDxoxn,
 MGL_R3_SDPnoa,
 MGL_R3_SDPSxoxn,
 MGL_R3_SSDxPDxax,
 MGL_R3_PDSanan,
 MGL_R3_PDSxna,
 MGL_R3_SDPSnoaxn,
 MGL_R3_DPSDpoaxx,
 MGL_R3_SPDaxn,
 MGL_R3_PSDPSoaxx,
 MGL_R3_DPSaxn,
 MGL_R3_DPSxx,

MGL_R3_PSDPSonoxx,
 MGL_R3_SDPSONoxn,
 MGL_R3_DSxn,
 MGL_R3_DPSnax,
 MGL_R3_SDPSoaxn,
 MGL_R3_SPDnax,
 MGL_R3_DSPDoaxn,
 MGL_R3_DSPDSaoxx,
 MGL_R3_PDSxan,
 MGL_R3_DPa,
 MGL_R3_PDSPnaoxn,
 MGL_R3_DPSnoa,
 MGL_R3_DSPDxoxn,
 MGL_R3_PDSPonoxn,
 MGL_R3_PDxn,
 MGL_R3_DSPnax,
 MGL_R3_PDSPoaxn,
 MGL_R3_DPSoa,
 MGL_R3_DPSoxn,
 MGL_R3_D,
 MGL_R3_DPSono,
 MGL_R3_SPDSxax,
 MGL_R3_DSPDaoxn,
 MGL_R3_DSPnao,
 MGL_R3_DPno,
 MGL_R3_PDSnoa,
 MGL_R3_PDSPxoxn,
 MGL_R3_SSPxDSxox,
 MGL_R3_SDPanan,
 MGL_R3_PSDnax,
 MGL_R3_DSPDoaxn,
 MGL_R3_DSPDPaoxx,
 MGL_R3_SDPxan,
 MGL_R3_PSDPxax,
 MGL_R3_DSPDaoxn,
 MGL_R3_DPSnao,
 MGL_R3_DSno,
 MGL_R3_SPDSanax,
 MGL_R3_SDxPDxan,
 MGL_R3_DPSxo,
 MGL_R3_DPSano,
 MGL_R3_PSa,
 MGL_R3_SPDSnaoxn,
 MGL_R3_SPDSonoxn,
 MGL_R3_Psxn,
 MGL_R3_SPDnoa,
 MGL_R3_SPDSxoxn,
 MGL_R3_SDPnax,
 MGL_R3_PSDPoaxn,
 MGL_R3_SDPoa,
 MGL_R3_SPDoxn,
 MGL_R3_DSPDxax,

MGL_R3_SPDSaoxn,
 MGL_R3_S,
 MGL_R3_SDPono,
 MGL_R3_SDPnao,
 MGL_R3_SPno,
 MGL_R3_PSDnoa,
 MGL_R3_PSDPxoxn,
 MGL_R3_PDSnax,
 MGL_R3_SPDSsoaxn,
 MGL_R3_SSPxPDxax,
 MGL_R3_DPSanan,
 MGL_R3_PSDPSaoxx,
 MGL_R3_DPSxan,
 MGL_R3_PDSPxax,
 MGL_R3_SDPSaoxn,
 MGL_R3_DPSDanax,
 MGL_R3_SPxDSxan,
 MGL_R3_SPDnao,
 MGL_R3_SDno,
 MGL_R3_SDPxo,
 MGL_R3_SDPano,
 MGL_R3_PDSoa,
 MGL_R3_PDSoxn,
 MGL_R3_DSPDxax,
 MGL_R3_PSDPaoxn,
 MGL_R3_SDPSxax,
 MGL_R3_PDSPaoxn,
 MGL_R3_SDPSanax,
 MGL_R3_SPxPDxan,
 MGL_R3_SSPxDSxax,
 MGL_R3_DSPDSanaxxn,
 MGL_R3_DPSao,
 MGL_R3_DPSxno,
 MGL_R3_SDPao,
 MGL_R3_SDPxno,
 MGL_R3_DSo,
 MGL_R3_SDPnoo,
 MGL_R3_P,
 MGL_R3_PDSono,
 MGL_R3_PDSnao,
 MGL_R3_PSnno,
 MGL_R3_PSDnao,
 MGL_R3_PDno,
 MGL_R3_PDSxo,
 MGL_R3_PDSano,
 MGL_R3_PDSao,
 MGL_R3_PDSxno,
 MGL_R3_DPo,
 MGL_R3_DPSnoo,
 MGL_R3_PSo,
 MGL_R3_PSDnoo,
 MGL_R3_DPSoo,


```

MGL_R3_1
} MGL_rop3CodesType

```

Prototype In mgraph.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.

To understand the naming of the ROP3 code, you need to understand how the source (S), destination (D) and pattern (P) pixels are combined into the final resulting destination pixel value. The ROP3 names essentially describe the resulting operation code in a reverse polish notation. Hence if the code is DPna, it means that the destination pixel is pushed on the computation stack. Then the pattern pixel is taken, it is inverted (Pn) and finally the inverted pattern is AND'ed with the destination pixel value. Likewise SDPxnon means that the destination pixel and pattern pixel are XOR'ed together (DPx), the result is then inverted (DPxn) which is then OR'ed with the source pixel data (SDPxno) and the result is then finally inverted again (SDPxnon).

Note: *We don't list the codes here for brevity as the ROP3 code names are self explanatory.*

MGL_stereoBufType

Declaration

```
typedef enum {  
    MGL_LEFT_BUFFER          = 0x0000,  
    MGL_RIGHT_BUFFER         = 0x8000  
} MGL_stereoBufType
```

Prototype In

mgraph.h

Description

Defines the flags passed to *MGL_setActivePage* to let the MGL know which buffer you wish to draw to when running in stereo mode (ie: after a display device context created with *MGL_createStereoDisplayDC*). This value is logical 'or'ed with the page parameter to *MGL_setActivePage*.

Members

<i>MGL_LEFT_BUFFER</i>	Draw to the left buffer in stereo modes
<i>MGL_RIGHT_BUFFER</i>	Draw to the right buffer in stereo modes

MGL_surfaceAccessFlagsType

Declaration

```
typedef enum {
    MGL_NO_ACCESS          = 0x0,
    MGL_VIRTUAL_ACCESS     = 0x1,
    MGL_LINEAR_ACCESS      = 0x2,
    MGL_SURFACE_FLAGS      = 0x3,
    MGL_STEREO_ACCESS      = 0x4,
    MGL_SHADOW_BUFFER      = 0x8
} MGL_surfaceAccessFlagsType
```

Prototype In

mgraph.h

Description

Defines the flags for the types of direct surface access provided.

Members

<i>MGL_NO_ACCESS</i>	Surface cannot be accessed
<i>MGL_VIRTUAL_ACCESS</i>	Surface is virtualised
<i>MGL_LINEAR_ACCESS</i>	Surface can be linearly accessed
<i>MGL_STEREO_ACCESS</i>	Surface supports stereo rendering
<i>MGL_SHADOW_BUFFER</i>	Display is using a system memory shadow buffer

MGL_suspendAppCodesType

Declaration

```
typedef enum {
    MGL_SUSPEND_APP      = 0,
    MGL_NO_SUSPEND_APP   = 1
} MGL_suspendAppCodesType
```

Prototype In

mgraph.h

Description

Defines the return codes that the application can return from the suspend application callback registered with the MGL. The default value to be returned is

MGL_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 95 and Windows NT in minimised form.

MGL_NO_SUSPEND_APP can be used to tell the MGL 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 MGL 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 which will always cause a switch away from the application immediately and force the surfaces to be lost.*

Members

MGL_SUSPEND_APP	Suspend application execution until restored
MGL_NO_SUSPEND_APP	Don't suspend execution, but allow switch

MGL_suspendAppFlagsType

Declaration

```
typedef enum {
    MGL_DEACTIVATE    = 0x0001,
    MGL_REACTIVATE    = 0x0002
} MGL_suspendAppFlagsType
```

Prototype In

mgraph.h

Description

Defines the suspend application callback flags, passed to the suspend application callback registered with the MGL. This callback is called when the user presses one of the system key sequences indicating that they wish to change the active application. The MGL 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. The MGL takes care of all the details about saving and restoring the state of the hardware, and all your application needs to do is save its own state so that you can re-draw the application screen upon re-activation.

Note: *Your application suspend callback may get called twice with the MGL_DEACTIVATE flag in order to test whether the switch should occur (under both DirectX and WinDirect fullscreen modes).*

Note: *When your callback is called with the MGL_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

MGL_DEACTIVATE	Application losing active focus
MGL_REACTIVATE	Application regaining active focus

MGL_textDirType

Declaration

```
typedef enum {  
    MGL_LEFT_DIR      = 0,  
    MGL_UP_DIR        = 1,  
    MGL_RIGHT_DIR     = 2,  
    MGL_DOWN_DIR      = 3  
} MGL_textDirType
```

Prototype In

mgraph.h

Description

Defines the text direction styles passed to *MGL_setTextDirection*

Members

<i>MGL_LEFT_DIR</i>	Text goes to left
<i>MGL_UP_DIR</i>	Text goes up
<i>MGL_RIGHT_DIR</i>	Text goes right
<i>MGL_DOWN_DIR</i>	Text goes down

MGL_textEncodingType

Declaration

```
typedef enum {
    MGL_ENCODING_ASCII          =    0,
    MGL_ENCODING_ISO8859_1,
    MGL_ENCODING_ISO8859_2,
    MGL_ENCODING_ISO8859_3,
    MGL_ENCODING_ISO8859_4,
    MGL_ENCODING_ISO8859_5,
    MGL_ENCODING_ISO8859_6,
    MGL_ENCODING_ISO8859_7,
    MGL_ENCODING_ISO8859_8,
    MGL_ENCODING_ISO8859_9,
    MGL_ENCODING_ISO8859_10,
    MGL_ENCODING_ISO8859_11,
    MGL_ENCODING_ISO8859_12,
    MGL_ENCODING_ISO8859_13,
    MGL_ENCODING_ISO8859_14,
    MGL_ENCODING_ISO8859_15,
    MGL_ENCODING_CP1250,
    MGL_ENCODING_CP1251,
    MGL_ENCODING_CP1252,
    MGL_ENCODING_CP1253,
    MGL_ENCODING_CP1254,
    MGL_ENCODING_CP1255,
    MGL_ENCODING_CP1256,
    MGL_ENCODING_CP1257,
    MGL_ENCODING_KOI8
} MGL_textEncodingType
```

Prototype In

mgraph.h

Description

Defines the text encoding (charset) as passed to *MGL_setTextEncoding*.

Members

<i>MGL_ENCODING_ASCII</i>	7bit ASCII (default)
<i>MGL_ENCODING_ISO8859_1</i>	Western European (ISO-8859-1)
<i>MGL_ENCODING_ISO8859_2</i>	Central European (ISO-8859-2)
<i>MGL_ENCODING_ISO8859_3</i>	Esperanto (ISO-8859-3)
<i>MGL_ENCODING_ISO8859_4</i>	Baltic (old) (ISO-8859-4)
<i>MGL_ENCODING_ISO8859_5</i>	Cyrillic (ISO-8859-5)
<i>MGL_ENCODING_ISO8859_6</i>	Arabic (ISO-8859-6)
<i>MGL_ENCODING_ISO8859_7</i>	Greek (ISO-8859-7)
<i>MGL_ENCODING_ISO8859_8</i>	Hebrew (ISO-8859-8)
<i>MGL_ENCODING_ISO8859_9</i>	Turkish (ISO-8859-9)
<i>MGL_ENCODING_ISO8859_10</i>	Nordic (ISO-8859-10)

<i>MGL_ENCODING_ISO8859_11</i>	Thai (ISO-8859-11) (not supported)
<i>MGL_ENCODING_ISO8859_12</i>	Indian (ISO-8859-12) (not supported)
<i>MGL_ENCODING_ISO8859_13</i>	Baltic (ISO-8859-13)
<i>MGL_ENCODING_ISO8859_14</i>	Celtic (ISO-8859-14)
<i>MGL_ENCODING_ISO8859_15</i>	Western European with Euro (ISO-8859-15)
<i>MGL_ENCODING_CP1250</i>	Windows Central European (CP 1250)
<i>MGL_ENCODING_CP1251</i>	Windows Cyrillic (CP 1251)
<i>MGL_ENCODING_CP1252</i>	Windows Western European (CP 1252)
<i>MGL_ENCODING_CP1253</i>	Windows Greek (CP 1253)
<i>MGL_ENCODING_CP1254</i>	Windows Turkish (CP 1254)
<i>MGL_ENCODING_CP1255</i>	Windows Hebrew (CP 1255)
<i>MGL_ENCODING_CP1256</i>	Windows Arabic (CP 1256)
<i>MGL_ENCODING_CP1257</i>	Windows Baltic (CP 1257)
<i>MGL_ENCODING_KOI8</i>	Russian KOI8-R

MGL_textJustType

Declaration

```
typedef enum {
    MGL_LEFT_TEXT      = 0,
    MGL_TOP_TEXT       = 0,
    MGL_CENTER_TEXT    = 1,
    MGL_RIGHT_TEXT     = 2,
    MGL_BOTTOM_TEXT    = 2,
    MGL_BASELINE_TEXT  = 3
} MGL_textJustType
```

Prototype In

mgraph.h

Description

Defines the text justification styles passed to *MGL_setTextJustify*.

Members

<i>MGL_LEFT_TEXT</i>	Justify from left
<i>MGL_TOP_TEXT</i>	Justify from top
<i>MGL_CENTER_TEXT</i>	Center the text
<i>MGL_RIGHT_TEXT</i>	Justify from right
<i>MGL_BOTTOM_TEXT</i>	Justify from bottom
<i>MGL_BASELINE_TEXT</i>	Justify from the baseline

MGL_waitVRTFlagType

Declaration

```
typedef enum {
    MGL_tripleBuffer    = 0,
    MGL_waitVRT         = 1,
    MGL_dontWait        = 2
} MGL_waitVRTFlagType
```

Prototype In

mgraph.h

Description

Defines for waitVRT flag for *MGL_setVisualPage*, *MGL_swapBuffers* and *MGL_glSwapBuffers*.

Members

<i>MGL_tripleBuffer</i>	Triple buffer. This mode enables hardware or software triple buffering if available on the target system. In this case when triple buffering is available the MGL will ensure that there is no flicker when flipping pages, however your frame rate will run at the maximum rate until you get to the physical refresh rate of the screen (ie: 60fps or higher). Note that if there is no hardware or software triple buffering available, this function will work like regular double buffering. Note also that you <i>must</i> have at least 3 pages available for triple buffering to work.
<i>MGL_waitVRT</i>	Wait for vertical retrace. This mode always waits for the vertical retrace when swapping display pages, and is required if only have two pages available to avoid flicker during animation.
<i>MGL_dontWait</i>	Don't wait for retrace. This mode simply programs the display start address change and returns. This may cause flicker on the screen during animation, and is mostly useful for debugging and testing purposes to see what the raw framerate of an animation is. Also if you dont have hardware or software triple available, and you allocate at least 3 pages you can achieve the effect of triple buffering if you know that the frame rate of your animation will not exceed the refresh rate of the screen.

MGL_wmWindowFlags

Declaration

```
typedef enum {
    MGL_WM_ALWAYS_ON_TOP          = 0x00000001,
    MGL_WM_ALWAYS_ON_BOTTOM      = 0x00000002,
    MGL_WM_FULL_REPAINT_ON_RESIZE = 0x00000004
} MGL_wmWindowFlags
```

Prototype In

mgraph.h

Description

Defines the flags passed to *MGL_wmSetWindowFlags* to specify special behavior of window.

Note: *MGL_WM_ALWAYS_ON_TOP* and *MGL_WM_ALWAYS_ON_BOTTOM* are mutually exclusive flags. If they are set both at the same time, the latter is ignored.

Members

<i>MGL_WM_ALWAYS_ON_TOP</i>	Window will stay on top of all its siblings
<i>MGL_WM_ALWAYS_ON_BOTTOM</i>	Window will stay under all its siblings
<i>MGL_WM_FULL_REPAINT_ON_RESIZE</i>	Window will repaint completely when resized. If this flag is not set, only the area that wasn't covered by the window before resize is repainted.

MGL_writeModeType

Declaration

```
typedef enum {
    MGL_R2_BLACK,
    MGL_R2_NOTMERGESRC,
    MGL_R2_MASKNOTSRC,
    MGL_R2_NOTCOPYSRC,
    MGL_R2_MASKSRCNOT,
    MGL_R2_NOT,
    MGL_R2_XORSRC,
    MGL_R2_NOTMASKSRC,
    MGL_R2_MASKSRC,
    MGL_R2_NOTXORSRC,
    MGL_R2_NOP,
    MGL_R2_MERGENOTSRC,
    MGL_R2_COPYSRC,
    MGL_R2_MERGESRCNOT,
    MGL_R2_MERGESRC,
    MGL_R2_WHITE,
    MGL_REPLACE_MODE    = MGL_R2_COPYSRC,
    MGL_AND_MODE        = MGL_R2_MASKSRC,
    MGL_OR_MODE         = MGL_R2_MERGESRC,
    MGL_XOR_MODE        = MGL_R2_XORSRC
} MGL_writeModeType
```

Prototype In

mgraph.h

Description

Defines the logical write mode operation codes for all drawing functions. The set of mix codes is the standard Microsoft Raster Operation (ROP2) codes between two values. We define the MGL 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.

Members

<i>MGL_R2_BLACK</i>	0
<i>MGL_R2_NOTMERGESRC</i>	DSon
<i>MGL_R2_MASKNOTSRC</i>	DSna
<i>MGL_R2_NOTCOPYSRC</i>	Sn
<i>MGL_R2_MASKSRCNOT</i>	SDna
<i>MGL_R2_NOT</i>	Dn
<i>MGL_R2_XORSRC</i>	DSx
<i>MGL_R2_NOTMASKSRC</i>	DSan
<i>MGL_R2_MASKSRC</i>	DSa
<i>MGL_R2_NOTXORSRC</i>	DSxn
<i>MGL_R2_NOP</i>	D

<i>MGL_R2_MERGENOTSRC</i>	DSno
<i>MGL_R2_COPYSRC</i>	S
<i>MGL_R2_MERGESRCNOT</i>	SDno
<i>MGL_R2_MERGESRC</i>	DSo
<i>MGL_R2_WHITE</i>	1
<i>MGL_REPLACE_MODE</i>	Replace mode
<i>MGL_AND_MODE</i>	AND mode
<i>MGL_OR_MODE</i>	OR mode
<i>MGL_XOR_MODE</i>	XOR mode

M_int16

Declaration

```
typedef short M_int16
```

Prototype In

mgraph.h

Description

Type definition for 16-bit signed values used in MGL.

M_int32

Declaration

```
typedef long M_int32
```

Prototype In

mgraph.h

Description

Type definition for 32-bit signed values used in MGL.

M_int8

Declaration

```
typedef char M_int8
```

Prototype In

mgraph.h

Description

Type definition for 8-bit signed values used in MGL.

M_uint16

Declaration

```
typedef unsigned short M_uint16
```

Prototype In

mgraph.h

Description

Type definition for 16-bit unsigned values used in MGL.

M_uint32

Declaration

```
typedef unsigned long M_uint32
```

Prototype In

mgraph.h

Description

Type definition for 32-bit unsigned values used in MGL.

M_uint8

Declaration

```
typedef unsigned char M_uint8
```

Prototype In

mgraph.h

Description

Type definition for 8-bit unsigned values used in MGL.

arc_coords_t

Declaration

```
typedef struct {
    int      x,y;
    int      startX,startY;
    int      endX,endY;
} arc_coords_t
```

Prototype In

mgraph.h

Description

Structure used to return elliptical arc starting and ending coordinates. This structure is used to obtain the exact center, starting and ending coordinates after an elliptical arc has been rasterized, so that you can properly turn the arc into a pie slice for example.

Members

<i>x</i>	x coordinate of the center of the elliptical arc
<i>y</i>	y coordinate of the center of the elliptical arc
<i>startX</i>	x coordinate of the starting pixel on the elliptical arc
<i>startY</i>	y coordinate of the starting pixel on the elliptical arc
<i>endX</i>	x coordinate of the ending pixel on the elliptical arc
<i>endY</i>	y coordinate of the ending pixel on the elliptical arc

attributes_t

Declaration

```
typedef struct {
    color_t      color;
    color_t      backColor;
    color_t      aaColor[5];
    int          backMode;
    color_t      bdrBright;
    color_t      bdrDark;
    point_t      CP;
    int          writeMode;
    int          penStyle;
    int          penHeight;
    int          penWidth;
    pattern_t     penPat[8];
    pixpattern_t  penPixPat[8];
    int          cntPenPat;
    int          cntPenPixPat;
    color_t      penPixPatTrans;
    int          lineStyle;
    uint         lineStipple;
    uint         stippleCount;
    int          polyType;
    int          fontBlendMode;
    int          srcBlendFunc;
    int          dstBlendFunc;
    uchar        alphaValue;
    ulong        planeMask;
    int          ditherMode;
    text_settings_t ts;
} attributes_t
```

Prototype In mgraph.h

Description

Structure representing the current MGL rendering attributes. This structure groups all of the MGL rendering state variables, and can be used to save and restore the entire MGL rendering state for any device context as a single unit.

Note: *You should only save and restore the state to the same device context!*

Members

<i>color</i>	Current foreground color
<i>backColor</i>	Current background color
<i>aaColor</i>	Palette for font anti-aliasing. Values correspond to 25/75, 50/50, and 75/25 percent blend of background and foreground colors respectively.

<i>backMode</i>	Background color mode for monochrome bitmap expansion. Will be either MGL_TRANSPARENT_BACKGROUND or MGL_OPAQUE_BACKGROUND, and determines how background pixels are drawn for monochrome bitmap rendering functions (including text).
<i>CP</i>	Current Position coordinate
<i>writeMode</i>	Current write mode. Will be one of the values defined by the <i>MGL_writeModeType</i> enumeration.
<i>penStyle</i>	Current pen fill style. Will be one of values defined by the <i>MGL_penStyleType</i> enumeration.
<i>penHeight</i>	Current pen height
<i>penWidth</i>	Current pen width
<i>penPat</i>	Array of 8 pen 8x8 monochrome bitmap pattern
<i>penPixPat</i>	Array of 8 pen 8x8 color pixmap pattern
<i>cntPenPat</i>	Current pen 8x8 monochrome bitmap pattern index (0-7)
<i>cntPenPixPat</i>	Current pen 8x8 color pixmap pattern index (0-7)
<i>penPixPatTrans</i>	Current transparent color for pixmap patterns
<i>lineStyle</i>	Current line style. Will be one of the values defined by the <i>MGL_lineStyleType</i> enumeration.
<i>lineStipple</i>	Current 16-bit line stipple mask.
<i>stippleCount</i>	Current line stipple counter.
<i>polyType</i>	Current polygon rasterizing type. Will be one of the values defined by the <i>MGL_polygonType</i> enumeration.
<i>fontBlendMode</i>	Type of blending used for anti-aliased fonts. Will be a value from the enumeration <i>MGL_fontBlendType</i> .
<i>srcBlendFunc</i>	Current src alpha blend function. Will be a value from the enumeration <i>MGL_blendFuncType</i> .
<i>dstBlendFunc</i>	Current dest alpha blend function. Will be a value from the enumeration <i>MGL_blendFuncType</i> .
<i>alphaValue</i>	Current constant alpha value between 0 and 255.
<i>planeMask</i>	Current plane mask to determine which bits get updated.
<i>ditherMode</i>	Current dither mode for blitting RGB bitmaps
<i>ts</i>	Current text drawing attributes

bitmap_t

Declaration

```
struct bitmap_t {
    int        width;
    int        height;
    int        bitsPerPixel;
    int        bytesPerLine;
    void       *surface;
    palette_t  *pal;
    pixel_format_t *pf;

}
```

Prototype In mgraph.h

Description

Structure representing a loaded lightweight bitmap image. This is the structure of Windows .BMP files after they have been loaded from disk with the *MGL_loadBitmap* function. Lightweight bitmaps have very little memory overhead once loaded from disk, since the entire bitmap information is stored in a single contiguous block of memory (although this is not necessary; see below). However the only thing you can do with a lightweight bitmap is display it to any MGL device context, using either stretching or transparency (*MGL_putBitmap*, *MGL_stretchBitmap*, *MGL_putSrcBitmapTransparent*). If you need to be able to draw on the bitmap surface, then you should load the bitmap into an MGL memory device context where you can call any of the standard MGL drawing functions and BitBlt operations on the bitmap. The only disadvantage of doing this is that a memory device context has a lot more memory overhead involved in maintaining the device context information.

You can build your own lightweight bitmap loading routines by creating the proper header information and loading the bitmap information into this structure. Note that although the MGL loads the bitmap files from disk with the bitmap surface, pixel format information and palette information all loaded into a single memory block, this is not necessary. If you wish you can create your own lightweight bitmaps with the bitmap surface allocated in a separate memory block and then use this bitmap header to blast information from this memory block to a device context as fast as possible.

Members

<i>width</i>	Width of the bitmap in pixels
<i>height</i>	Height of the bitmap in pixels
<i>bitsPerPixel</i>	Pixel depth of the bitmap
<i>bytesPerLine</i>	Scanline width for the bitmap. The scanline width must always be aligned to a DWORD boundary, so the minimum scanline width is 4 bytes.
<i>surface</i>	Pointer to the bitmap surface.
<i>pal</i>	Pointer to the bitmap palette. If this field is NULL, the

pf

bitmap does not have an associated palette.
Pointer to the bitmap pixel format info. This field will be NULL for all bitmaps with 8 or less bits per pixel, but will always be properly filled in for bitmaps with 15 or more bits per pixel.

bltfx_t

Declaration

```
typedef struct {
    ulong        flags;
    int          writeMode;
    color_t      colorKeyLo;
    color_t      colorKeyHi;
    int          srcBlendFunc;
    int          dstBlendFunc;
    color_t      constColor;
    int          constAlpha;
} bltfx_t
```

Prototype In

mgraph.h

Description

Information structure passed to the *MGL_bitBltFx* and related functions. This structure defines the type of BitBlt operation that is performed by the *MGL_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.

If write mode is enabled, the writeMode member is used to determine the logical write mode operation for combining pixels with the destination surface. If write mode is not enabled, MGL_REPLACE_MODE is assumed.

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. Color keying is used for implementing transparent blits in both source and destination transparency. Note also that if color keying is enabled with color conversion, the colorKeyLo and colorKeyHi values **must** be in the same format as the destination surface, not the source surface. For instance if you are color converting an 8-bit bitmap to a 32-bit display DC and wish to use transparency, the colorKeyLo and colorKeyHi values must be 32-bit RGB values and not 8-bit color index values.

If blending is enabled, the srcBlendFunc, dstBlendFunc, constColor and and constAlpha values are used to implement the blending operation. Unlike all other drawing functions, the *MGL_bitBltFx* family of functions do not honor the global MGL blending codes set by MGL_setBlendingFunc, but instead control blending directly via the blending flags in this structure. Also note that enabling any blending operation overrides the setting of the supplied write mode operation. Logical write modes and blending cannot be used at the same time.

Members

<i>flags</i>	Flags to define the type of BitBlt operation (<i>MGL_bitBltFxFlagsType</i>)
--------------	--

<i>writeMode</i>	Logical write mode operation (if write mode is 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>MGL_blendFuncType</i>)
<i>dstBlendFunc</i>	Dst blend function (<i>MGL_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)

captureentry_t

Declaration

```
struct captureentry_t {
    ulong                mask;
    struct window_t      *wnd;
    int                  id;
    struct captureentry_t *next;
}
```

Prototype In

mgraph.h

Description

Internal structure describing how to route events within window manager. By default, all events go to the window under mouse pointer. *captureentry_t* provides means to override this and redirect certain (user-specified) events to particular windows.

Members

<i>mask</i>	Mask specifying type of events
<i>hndFunc</i>	Window that will receive these events
<i>id</i>	User-defined identifier of capture entry (used for entry removal)
<i>next</i>	Pointer to next capture entry in the chain

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.

color16_cursor_t

Declaration

```
typedef struct {
    ulong      bitsPerPixel;
    uchar      colorData[2048];
    uchar      andMask[512];
    palette_t   palette[16];
    ulong      xHotSpot;
    ulong      yHotSpot;
} color16_cursor_t
```

Prototype In mgraph.h

Description

Structure representing a 16-color color mouse cursor. The cursor is defined as a 64x64 image with an AND and XOR mask. 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	00	Transparent (color from screen memory)
0	0F	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 or 0F to either make the pixel transparent or the inversion of the screen pixel. Any other value will produce an undefined result.

The xHotSpot and yHotSpot 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.

Note that Microsoft Windows cursors stored on disk are generally 32x32 in size, and the MGL will convert them during loading to the 64x64 internal format. The MGL does however support loading 64x64 cursors, but you may need to manually create these yourself as Windows resource editors appear to be hard coded to use 32x32 cursors.

Members

<i>bitsPerPixel</i>	Indicates the number of bits per pixel for cursor (4 in this case)
<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>xHotSpot</i>	x coordinate of the mouse hotspot location. The mouse hotspot location is used to properly align the mouse cursor image to the actual mouse cursor position on the screen

yHotSpot y coordinate of the mouse hotspot location

color256_cursor_t

Declaration

```
typedef struct {
    ulong    bitsPerPixel;
    uchar    colorData[4096];
    uchar    andMask[512];
    palette_t palette[256];
    ulong    xHotSpot;
    ulong    yHotSpot;
} color256_cursor_t
```

Prototype In mgraph.h

Description

Structure representing a 256-color color mouse cursor. The cursor is defined as a 64x64 image with an AND and XOR mask. 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	00	Transparent (color from screen memory)
0	FF	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 or FF to either make the pixel transparent or the inversion of the screen pixel. Any other value will produce an undefined result.

The xHotSpot and yHotSpot 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.

Note that Microsoft Windows cursors stored on disk are generally 32x32 in size, and the MGL will convert them during loading to the 64x64 internal format. The MGL does however support loading 64x64 cursors, but you may need to manually create these yourself as Windows resource editors appear to be hard coded to use 32x32 cursors.

Members

<i>bitsPerPixel</i>	Indicates the number of bits per pixel for cursor (8 in this case)
<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>xHotSpot</i>	x coordinate of the mouse hotspot location. The mouse hotspot location is used to properly align the mouse cursor image to the actual mouse cursor position on the screen

yHotSpot y coordinate of the mouse hotspot location

colorRGBA_cursor_t

Declaration

```
typedef struct {
    ulong        bitsPerPixel;
    uchar        colorData[16384];
    ulong        xHotSpot;
    ulong        yHotSpot;
} colorRGBA_cursor_t
```

Prototype In

mgraph.h

Description

Hardware 32-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 32-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.

Structure representing a 32-bit RGBA alpha blended color mouse cursor. The cursor is defined as a 64x64 32-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 xHotSpot and yHotSpot 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.

Note that Microsoft Windows cursors stored on disk are generally 32x32 in size, and the MGL will convert them during loading to the 64x64 internal format. The MGL does however support loading 64x64 cursors, but you may need to manually create these yourself as Windows resource editors appear to be hard coded to use 32x32 cursors.

Members

<i>bitsPerPixel</i>	Indicates the number of bits per pixel for cursor (32 in this case)
<i>colorData</i>	Cursor color data as a 64x64 array of packed 32-bit pixels
<i>xHotSpot</i>	x coordinate of the mouse hotspot location. The mouse hotspot location is used to properly align the mouse cursor image to the actual mouse cursor position on the screen
<i>yHotSpot</i>	y coordinate of the mouse hotspot location

colorRGB_cursor_t

Declaration

```
typedef struct {
    ulong    bitsPerPixel;
    uchar    colorData[12288];
    uchar    andMask[512];
    ulong    xHotSpot;
    ulong    yHotSpot;
} colorRGB_cursor_t
```

Prototype In mgraph.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.

Note that Microsoft Windows cursors stored on disk are generally 32x32 in size, and the MGL will convert them during loading to the 64x64 internal format. The MGL does however support loading 64x64 cursors, but you may need to manually create these yourself as Windows resource editors appear to be hard coded to use 32x32 cursors.

Members

<i>bitsPerPixel</i>	Indicates the number of bits per pixel for cursor (24 in this case)
<i>colorData</i>	Cursor color data as a 64x64 array of packed 24-bit pixels
<i>andMask</i>	Cursor AND mask
<i>xHotSpot</i>	x coordinate of the mouse hotspot location. The mouse hotspot location is used to properly align the mouse cursor image to the actual mouse cursor position on the

yHotSpot screen
y coordinate of the mouse hotspot location

color_t**Declaration**

```
typedef unsigned long color_t
```

Prototype In

mgraph.h

Description

Type definition for all color values used in MGL. All color values are 32 bits wide, and can be either a 4 or 8 bit color index, or a packed RGB tuple depending on the pixel format for the display mode. For packed RGB display modes, the colors may contain 15, 16, 24 or 32 bits of color information, and the format of the RGB colors is stored in the *pixel_format_t* structure. You should use the *MGL_packColor* family of functions to encode color values in RGB modes, and use the *MGL_unpackColor* family of functions to extract color values in RGB modes.

cursor_t

Declaration

```
typedef union {
    mono_cursor_t      m;
    color16_cursor_t   c16;
    color256_cursor_t  c256;
    colorRGB_cursor_t  cRGB;
    colorRGBA_cursor_t cRGBA;
} cursor_t
```

Prototype In

mgraph.h

Description

Structure representing a loaded mouse cursor. This is the structure of the mouse cursor data after it has been loaded from disk by the MGL, and is used to set the mouse cursor shape. You can build your own mouse cursors manually by filling in this structure.

Note that this structure is actually a union of different cursor structures, and you should examine the 'colors' member of the 'm' member to determine the number of colors in the cursor, and then use the appropriate union member to access the cursor image data directly. The size of the cursor image is dependent on the type of cursor stored in the structure.

Members

<i>m</i>	Structure for monochrome or 2-color cursors
<i>c16</i>	Structure for 4-bit cursors (16 colors)
<i>c256</i>	Structure for 8-bit cursors (256 colors)
<i>cRGB</i>	Structure for 24-bit RGB color cursors (16.7M colors)
<i>cRGBA</i>	Structure for 32-bit RGBA alpha blended cursors (16.7M colors)

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.

fileio_t

Declaration

```
typedef struct {
    FILE *    (*fopen) (const char *filename, const char *mode);
    int       (*fclose) (FILE *f);
    int       (*fseek) (FILE *f, long offset, int whence);
    long      (*ftell) (FILE *f);
    size_t    (*fread) (void *ptr, size_t size, size_t n, FILE *f);
    size_t    (*fwrite) (const void *ptr, size_t size, size_t n, FILE
*f);
} fileio_t
```

Prototype In

mgraph.h

Description

Structure representing the set of file I/O functions that can be overridden in the MGL. When you override the file I/O functions in the MGL, you must provide a compatible function for each of the entries in this structure that behave identically to the standard C library I/O functions of similar names.

Note: Once you have overridden the file I/O functions, you can access the overridden functions from other libraries and DLL's by calling the `MGL_fopen` family of functions, which are simply stubs to call the currently overridden function via the function pointers.

Members

<i>fopen</i>	Standard C fopen function replacement
<i>fclose</i>	Standard C fclose function replacement
<i>fseek</i>	Standard C fseek function replacement
<i>ftell</i>	Standard C ftell function replacement
<i>fread</i>	Standard C fread function replacement
<i>fwrite</i>	Standard C fwrite function replacement

fix32_t**Declaration**

```
typedef long                fix32_t
```

Prototype In

mgraph.h

Description

Type definition for all standard 32-bit fixed point values used in MGL. Standard fixed point values are 32-bits wide, and represented in 16.16 fixed point format (16 bits of integer, 16 bits of fraction). These numbers can represent signed numbers from +32767.9 to -32768.9.

Note: *If you are doing fixed point arithmetic for screen coordinate calculations, be very careful of overflow conditions when doing multiplication operations.*

font_info_t

Declaration

```
struct font_info_t {
    char        familyName[_MGL_FNAME_SIZE];
    short       fontLibType;
    ibool       isFixed;
    char        regularFace[256];
    char        boldFace[256];
    char        italicFace[256];
    char        boldItalicFace[256];
}
```

Prototype In

mgraph.h

Description

Structure representing information about one font family. Family usually contains several faces

Members

<i>familyName</i>	name of family, e.g. 'Arial'
<i>fontLibType</i>	Integer representing the type of font library file. Will be one of the values defined by the <i>MGL_fontLibType</i> enumeration.
<i>isFixed</i>	True if the font is fixed width, false if proportional font.
<i>regularFace</i>	Filename of font file with regular face. Pass this to <i>MGL_openFontLib</i> to load the font. Will be empty string if the family does not contain this face.
<i>boldFace</i>	Filename of font file with bold face. Pass this to <i>MGL_openFontLib</i> to load the font. Will be empty string if the family does not contain this face.
<i>italicFace</i>	Filename of font file with italic face. Pass this to <i>MGL_openFontLib</i> to load the font. Will be empty string if the family does not contain this face.
<i>boldItalicFace</i>	Filename of font file with regular face. Pass this to <i>MGL_openFontLib</i> to load the font. Will be empty string if the family does not contain this face.

font_lib_t

Declaration

```
struct font_lib_t {
    char        name[_MGL_FNAME_SIZE];
    short       fontLibType;
    FILE        *f;
    ibool       ownHandle;
    ulong       dwOffset;
}
```

Prototype In

mgraph.h

Description

Structure representing a loaded MGL font library. MGL font files come in three flavors, either vector fonts, bitmap fonts or scaleable fonts. Vector fonts represent the characters in the font as a set of lines that are drawn, and vector fonts can be scaled and rotated to any desired angle. Vector fonts however do not look very good when rasterized at high resolutions. Bitmap fonts represent the characters in the font as small monochrome bitmaps, and can be either fixed width fonts or proportional fonts. Scaleable fonts (TrueType and Adobe Type 1) represent the characters as mathematical outlines that can be scaled to any size. Scaleable fonts are scan converted into bitmap fonts at runtime when a particular font point size is requested. Scalable fonts can also be used for anti-aliased text rendering for better looking fonts at lower resolutions.

The MGL can load both MGL 1.x style font files (vector and bitmap fonts) or Windows 2.x style bitmap font files. For creating your own bitmap font files, you should use any standard Windows font file editor and save the fonts in Windows 2.x format (which is the standard format used by Windows 3.x, Windows 95 and Windows NT for bitmap fonts).

font_t**Declaration**

```
struct font_t {  
    char          name[_MGL_FNAME_SIZE];  
    short         fontType;  
    short         maxWidth;  
    short         maxKern;  
    short         fontWidth;  
    short         fontHeight;  
    short         ascent;  
    short         descent;  
    short         leading;  
    short         pointSize;  
}
```

Prototype In

mgraph.h

Description

Structure representing an MGL font ready for drawing. MGL fonts are loaded from disk in two flavors, either vector fonts or bitmap fonts. Vector fonts represent the characters in the font as a set of lines that are drawn, and vector fonts can be scaled and rotated to any desired angle. Vector fonts however do not look very good when rasterized at high resolutions. Bitmap fonts represent the characters in the font as small monochrome bitmaps, and can be either fixed width fonts or proportional fonts. Bitmap fonts are also what are generated when you load a TrueType or Adobe Type 1 scalable font library from disk.

fxpoint_t**Declaration**

```
typedef struct {  
    fix32_t x,y;  
} fxpoint_t
```

Prototype In

mgraph.h

Description

Structure describing a 16.16 fixed point coordinate.

Members

<i>x</i>	Fixed point x coordinate
<i>y</i>	Fixed point y coordinate

globalevententry_t

Declaration

```
struct globalevententry_t {
    ulong                mask;
    globaleventhandler_t hndFunc;
    int                 id;
    struct globalevententry_t *next;
}
```

Prototype In

mgraph.h

Description

Internal structure describing an entry in global event table for *winmng_t*. MGL window manager distributes events to global event handlers prior to sending them to windows. Event entries describe which events go to which handlers.

Members

<i>mask</i>	Mask specifying what types of events the handler accepts
<i>hndFunc</i>	Event handler callback
<i>id</i>	User-defined identifier of event entry (used for entry removal)
<i>next</i>	Pointer to next event entry in the chain

gmode_t

Declaration

```
typedef struct {
    int      xRes;
    int      yRes;
    int      bitsPerPixel;
    color_t  maxColor;
    int      maxPage;
    int      bytesPerLine;
    int      aspectRatio;
    long     pageSize;
    int      scratch1;
    int      scratch2;
    int      redMaskSize;
    int      redFieldPosition;
    int      greenMaskSize;
    int      greenFieldPosition;
    int      blueMaskSize;
    int      blueFieldPosition;
    int      alphaMaskSize;
    int      alphaFieldPosition;
    ulong    modeFlags;
    ulong    bitmapStartAlign;
    ulong    bitmapStridePad;
} gmode_t
```

Prototype In

mgraph.h

Description

Structure representing the attributes for a specific video mode. This structure is also used to store the rendering dimensions for all device context surfaces in the *MGLDC* structure.

Note that when the mode is a color index display mode, the *redMaskSize* and *redFieldPosition* hold the location and size of the color index value within the pixel.

Members

<i>xRes</i>	Device x resolution - 1
<i>yRes</i>	Device y resolution - 1
<i>bitsPerPixel</i>	Pixel depth
<i>maxColor</i>	Maximum color for device - 1
<i>maxPage</i>	Maximum number of hardware display pages - 1
<i>bytesPerLine</i>	Number of bytes in a single device scanline
<i>aspectRatio</i>	Device pixel aspect ratio ((horiz/vert) * 1000)
<i>pageSize</i>	Number of bytes in a hardware display page
<i>scratch1</i>	Internal scratch value
<i>scratch2</i>	Internal scratch value

<i>redMaskSize</i>	Size of RGB red mask (also color index)
<i>redFieldPosition</i>	Number of bits in RGB red field (also color index)
<i>greenMaskSize</i>	Size of RGB green mask
<i>greenFieldPosition</i>	Number of bits in RGB green field
<i>blueMaskSize</i>	Size of RGB blue mask
<i>blueFieldPosition</i>	Number of bits in RGB blue field
<i>alphaMaskSize</i>	Size of RGB alpha mask
<i>alphaFieldPosition</i>	Number of bits in RGB alpha field
<i>modeFlags</i>	Flags for the mode
<i>bitmapStartAlign</i>	Linear offscreen bitmap start alignment in bytes
<i>bitmapStridePad</i>	Linear offscreen bitmap stride pad in bytes

icon_t

Declaration

```
typedef struct {
    int        byteWidth;
    uchar      *andMask;
    bitmap_t   xorMask;

    } icon_t
```

Prototype In

mgraph.h

Description

Structure representing a loaded icon. Icons are used by the MGL to display small, transparent bitmap images that can be of any dimension. The standard Windows .ICO files can store icons in 32x32 and 64x64 formats, although the MGL can load icons of any dimensions if you can find an editor that will allow you to create large icons.

Icons are always drawn by the MGL by first using the icon AND mask to punch a hole in the background of the display surface, and then the icon bitmap XOR mask is XOR'ed into the display surface. This method is compatible with the way that Microsoft Windows displays icons on the screen.

Members

<i>byteWidth</i>	Width of the monochrome AND mask in bytes. Must be consistent with the bitmap width in the xorMask structure.
<i>andMask</i>	Pointer to the AND mask information, which is stored contiguously in memory after the header block. The dimensions of the AND mask is defined by the dimensions of the xorMask bitmap image.
<i>xorMask</i>	Bitmap image header block, containing information about the mask used to draw the icon image. The actual bitmap surface and palette data is stored contiguously in memory after the header block.

metrics_t

Declaration

```
typedef struct {
    int          width;
    int          fontWidth;
    int          fontHeight;
    int          ascent;
    int          descent;
    int          leading;
    int          kern;
} metrics_t
```

Prototype In

mgraph.h

Description

Structure representing text metrics for a font or a single character, in the current text attributes. For bitmap fonts you can get all the metric information from the *font_t* structure, however for vector fonts, this routine will provide the proper metrics for the font after being scaled by the current font character scaling size. This structure is also used to obtain specified 'tightest fit' metrics information about any character in the font.

Members

<i>width</i>	Actual width of the character in pixels
<i>fontWidth</i>	Font character width, including any extra padding between this character and the next character. This value is used to advance the current position to the start of the next character, and can be larger than the actual character width (in order to put space between the characters).
<i>fontHeight</i>	Standard height of the font (not including the leading value).
<i>ascent</i>	Font or character ascent value. The ascent value is the number of pixels that the font extends up from the font's baseline.
<i>descent</i>	Font or character descent value. The descent value is the number of pixels that the font extends down from the font's baseline.
<i>leading</i>	Font leading value. The leading value is the number of vertical pixels of space that are usually required between two lines of text drawn with this font.
<i>kern</i>	Character kern value. The kern value for the character is the number of pixels it extends back past the character origin (such as the tail of the lowercase j character for some fonts).

mono_cursor_t

Declaration

```
typedef struct {
    ulong    bitsPerPixel;
    uchar    xorMask[512];
    uchar    andMask[512];
    ulong    xHotSpot;
    ulong    yHotSpot;
} mono_cursor_t
```

Prototype In

mgraph.h

Description

Structure representing a monochrome or 2-color mouse cursor. 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 xHotSpot and yHotSpot 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.

Note that Microsoft Windows cursors stored on disk are generally 32x32 in size, and the MGL will convert them during loading to the 64x64 internal format. The MGL does however support loading 64x64 cursors, but you may need to manually create these yourself as Windows resource editors appear to be hard coded to use 32x32 cursors.

Members

<i>bitsPerPixel</i>	Indicates the number of bits per pixel for cursor (1 in this case)
<i>xorMask</i>	64x64 bit XOR pixel mask
<i>andMask</i>	64x64 bit AND pixel mask (see note above)
<i>xHotSpot</i>	x coordinate of the mouse hotspot location. The mouse hotspot location is used to properly align the mouse cursor image to the actual mouse cursor position on the screen
<i>yHotSpot</i>	y coordinate of the mouse hotspot location

palette_ext_t

Declaration

```
typedef struct {  
    ushort  blue;  
    ushort  green;  
    ushort  red;  
} palette_ext_t
```

Prototype In

mgraph.h

Description

Structure representing a single extended color palette entry. Extended color palette entries are similar to regular color palette entries, however the extended color palette supports 16 bits per primary, allowing for higher color resolution than a regular 8 bit primary color palette. The extended color palette is used to build the color lookup tables for all the extended color index device contexts used in the MGL.

Members

<i>blue</i>	Blue channel color (0 - 65535)
<i>green</i>	Green channel color (0 - 65535)
<i>red</i>	Red channel color (0 - 65535)

palette_t**Declaration**

```
struct palette_t {  
    uchar    blue;  
    uchar    green;  
    uchar    red;  
    uchar    alpha;  
}
```

Prototype In

mgraph.h

Description

Structure representing a single color palette entry. Color palette entries are used to build the color lookup tables for all the device contexts used in the MGL, which are used to define the final color values for colors in color index modes (8-bits per pixel and below). Color palette information is always stored in 8-bits per primary format (ie: 8-bits of red, green and blue information), and will be converted by MGL to the appropriate color format used by the underlying hardware when the hardware palette is programmed. Hence in standard VGA modes (which only use 6-bits per primary) the bottom two bits of color information will be lost when the palette is programmed.

Members

<i>blue</i>	Blue channel color (0 - 255)
<i>green</i>	Green channel color (0 - 255)
<i>red</i>	Red channel color (0 - 255)
<i>alpha</i>	Alignment value (not used and should always be 0)

pattern_t**Declaration**

```
typedef struct {  
    uchar    p[8];  
} pattern_t
```

Prototype In

mgraph.h

Description

Type definition for an 8x8 monochrome bitmap pattern. This is used to specify the monochrome bitmap pattern used for filling solid objects when the pen style is MGL_BITMAP_OPAQUE or MGL_BITMAP_TRANSPARENT.

When the pen style is MGL_BITMAP_OPAQUE, where bits in the pattern are a 1, the foreground color is used. Where bits in the pattern are a 0, the background color is used.

When the pen style is MGL_BITMAP_TRANSPARENT, where bits in the pattern are a 1, the foreground color is used. Where bits in the pattern are a 0, the pixel is left unmodified on the screen.

Members

p 8 bytes of pattern data

pixel_format_t

Declaration

```
typedef struct {
    uchar    redMask;
    uchar    redPos;
    uchar    redAdjust;
    uchar    greenMask;
    uchar    greenPos;
    uchar    greenAdjust;
    uchar    blueMask;
    uchar    bluePos;
    uchar    blueAdjust;
    uchar    alphaMask;
    uchar    alphaPos;
    uchar    alphaAdjust;
} pixel_format_t
```

Prototype In

mgraph.h

Description

Structure representing the format of an RGB pixel. This structure is used to describe the current RGB pixel format used by all MGL device contexts with 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:

24-bit	Description
RGB	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.
BGR	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:

32-bit	Description
ARGB	Values are packed with Red in byte 2, Green in byte 1 and Blue in byte 0 and alpha in byte 3.
ABGR	Values are packed with Blue in byte 2, Green in byte 1 and Red in byte 0 and alpha in byte 3.
RGBA	Values are packed with Red in byte 3, Green in byte 2 and Blue in byte 1 and alpha in byte 0.
BGRA	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 15-bits per pixel and higher 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. The MGL will perform pixel format translation on the fly for *MGL_bitBlt* operations, but this can be time consuming. The formula for packing the pixel data into the proper positions given three 8-bit RGB values is as follows:

```
color = ((color_t)((R >> redAdjust) & redMask)
        << redPos)
        | ((color_t)((G >> greenAdjust) & greenMask)
        << greenPos)
        | ((color_t)((B >> blueAdjust) & blueMask)
        << bluePos);
```

Alternatively you can unpack the color values with the following code:

```
R = (((color) >> redPos) & redMask)
    << redAdjust;
G = (((color) >> greenPos) & greenMask)
    << greenAdjust;
B = (((color) >> bluePos) & 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 MGL 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
```

Note: For 32-bit modes, the alpha channel information is unused, but should always be set to zero. Some hardware devices interpret the alpha channel information so unless you use a value of zero, you will get some strange looking results on the screen.

Members

redMask

Unshifted 8-bit mask for the red color channel, and will be

	5-bits wide for a 5-bit color channel or 8-bits wide for an 8-bit color channel.
<i>redPos</i>	Bit position for bit 0 of the red color channel information.
<i>redAdjust</i>	Number of bits to shift the 8 bit red value right
<i>greenMask</i>	Unshifted 8-bit mask for the green color channel.
<i>greenPos</i>	Bit position for bit 0 of the green color channel information.
<i>greenAdjust</i>	Number of bits to shift the 8 bit green value right
<i>blueMask</i>	Unshifted 8-bit mask for the blue color channel.
<i>bluePos</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>alphaPost</i>	Bit position for bit 0 of the alpha channel information
<i>alphaAdjust</i>	Number of bits to shift the 32 bit alpha value right

pixpattern16_t

Declaration

```
typedef struct {  
    ushort      p[8][8];  
} pixpattern16_t
```

Prototype In

mgraph.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

pixpattern24_t

Declaration

```
typedef struct {  
    uchar    p[8][8][3];  
} pixpattern24_t
```

Prototype In

mgraph.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

pixpattern32_t

Declaration

```
typedef struct {  
    ulong      p[8][8];  
} pixpattern32_t
```

Prototype In

mgraph.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

pixpattern8_t**Declaration**

```
typedef struct {  
    uchar    p[8][8];  
} pixpattern8_t
```

Prototype In

mgraph.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

pixpattern_t**Declaration**

```
typedef union {  
    pixpattern8_t    b8;  
    pixpattern16_t   b16;  
    pixpattern24_t   b24;  
    pixpattern32_t   b32;  
} pixpattern_t
```

Prototype In

mgraph.h

Description

Type definition for an 8x8 color pixmap pattern. This is used to specify the color pixmap pattern used for filling solid objects when the pen style is in MGL_PIXMAP mode. The pixmap pattern is defined as an 8x8 array of color values. 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 in the MGL device context that the pattern is loaded into.

point_t**Declaration**

```
typedef struct {  
    int x,y;  
} point_t
```

Prototype In

mgraph.h

Description

Structure describing an integer point passed to the MGL.

Members

<i>x</i>	X coordinate for the point
<i>y</i>	Y coordinate for the point

rect_t**Declaration**

```
typedef struct {  
    int left;  
    int top;  
    int right;  
    int bottom;  
} rect_t
```

Prototype In

mgraph.h

Description

Structure describing an integer rectangle. Note that MGL 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

region_t

Declaration

```
typedef struct {
    rect_t      rect;
    span_t      *spans;
} region_t
```

Prototype In

mgraph.h

Description

Structure 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 (such as the SciTech MegaVision Library which makes extensive use of the MGL's region management functions).

Members

<i>rect</i>	Bounding rectangle for the region. If the spans field below 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 <i>MGL_isSimpleRegion</i> function to determine if the region contains only a single rectangle.
<i>spans</i>	Pointer to the internal region span structure.

segment_t**Declaration**

```
struct segment_t {  
    struct segment_t    *next;  
    int                  x;  
}
```

Prototype In

mgraph.h

Description

Structure 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

span_t**Declaration**

```
struct span_t {  
    struct span_t    *next;  
    segment_t        *seg;  
    int              y;  
}
```

Prototype In

mgraph.h

Description

Structure 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

text_settings_t

Declaration

```
typedef struct {
    int        horizJust;
    int        vertJust;
    int        dir;
    int        szNumerx;
    int        szNumery;
    int        szDenomx;
    int        szDenomy;
    int        spaceExtra;
    font_t     *font;
    ushort     *encoding;
    ibool      useEncoding;
} text_settings_t
```

Prototype In

mgraph.h

Description

Structure representing the current text rasterizing settings. This structure is used to group all these settings together in the MGL, and allows you to save and restore the text rendering settings as a single unit.

Members

<i>horizJust</i>	Horizontal justification value. Will be one of the values defined by the <i>MGL_textJustType</i> enumeration.
<i>vertJust</i>	Vertical justification value. Will be one of the values defined by the <i>MGL_textJustType</i> enumeration.
<i>dir</i>	Current text direction value. Will be one of the values defined by the <i>MGL_textDirType</i> enumeration.
<i>szNumerx</i>	Current text x size numerator value
<i>szNumery</i>	Current text y size numerator value
<i>szDenomx</i>	Current text x size denominator value
<i>szDenomy</i>	Current text y size denominator value
<i>spaceExtra</i>	Current text space extra value. The space extra value is the number of extra pixels to be added to every space character when rendering the line of text.
<i>font</i>	Pointer to current active font loaded in memory.
<i>encoding</i>	Pointer to current encoding table.
<i>useEncoding</i>	Flag indication whether to use encoding table or not. Is true for TT and Type1 fonts, false for bitmap fonts

window_t

Declaration

```
struct window_t {
    int                x, y;
    size_t             width, height;
    winmng_t           *wm;
    struct window_t    *parent;
    struct window_t    *next, *prev;
    struct window_t    *firstChild, *lastChild;
    windowevententry_t *eventHandlers;
    void               *userData;
    windtor_t          dtor;
    painter_t           painter;
    cursor_t            *cursor;
    ibool               visible;
    long                flags;
}
```

Prototype In

mgraph.h

Description

Window is rectangular area of the screen managed by window manager. Window may contain unlimited number of child windows that are placed inside its area and may themselves contain children. Windows are drawn using a painter callback set with *MGL_wmSetWindowPainter* and MGL window manager guarantees that the painter never draws anything outside the window's area. Windows may be partially or fully covered by other windows, in which case MGL ensures proper clipping.

MGL contains powerful system for events propagation. Whenever user's action generates an event (e.g. mouse click), it is distributed to the window it belongs (e.g. the window under mouse pointer or the one that captured keyboard).

You can use *MGL_wmCreateWindow* to create windows.

Members

<i>x</i>	X coordinate of window position
<i>y</i>	Y coordinate of window position
<i>width</i>	Window width
<i>height</i>	Window height
<i>wm</i>	Pointer to window manager that owns the window
<i>parent</i>	Pointer to the parent window
<i>next</i>	Pointer to next sibling window
<i>prev</i>	Pointer to previous sibling window
<i>firstChild</i>	Pointer to the first child window
<i>lastChild</i>	Pointer to the last child window
<i>eventHandlers</i>	Table of event handlers for this window
<i>userData</i>	Arbitrary data pointer for user's needs

<i>dtor</i>	Destructor callback called from MGL_wxDestroyWindow
<i>painter</i>	Painter callback
<i>cursor</i>	Mouse cursor associated with the window
<i>visible</i>	Boolean flag indicating if the window is currently visible
<i>flags</i>	Combination of flags from <i>MGL_wmWindowFlags</i>

windowevententry_t

Declaration

```
struct windowevententry_t {
    ulong                mask;
    windoweventhandler_t hndFunc;
    int                 id;
    struct windowevententry_t *next;
}
```

Prototype In

mgraph.h

Description

Internal structure describing an entry in event table for *window_t*. MGL window manager distributes events to event handlers attached to windows. Event entries describe which events go to which handlers.

Members

<i>mask</i>	Mask specifying what types of events the handler accepts
<i>hndFunc</i>	Event handler callback
<i>id</i>	User-defined identifier of event entry (used for entry removal)
<i>next</i>	Pointer to next event entry in the chain

winmng_t

Declaration

```
typedef struct {
    globalevententry_t    *globalEventHandlers;
    captureentry_t        *capturedEvents;
    cursor_t              *globalCursor;
    struct window_t        *rootWnd;
    struct window_t        *activeWnd;
    MGLDC                 *dc;
    region_t               *invalidatedRegion;
    attributes_t           dcAttrs;
    ibool                  updatingDC;
} winmng_t
```

Prototype In

mgraph.h

Description

This structure represents MGL window manager. MGL WM provides functionality similar to that of Xlib, i.e. bare minimum needed to implement windowing environment on top of SciTech MGL. That is, it manages hierarchy of rectangular windows, takes care of proper repainting (but you must provide painter functions for all windows) and clipping and distributes input events among the windows.

You must create an instance of this object with *MGL_wmCreate* prior to using WM functionality and destroy it before shutting MGL down with *MGL_wmDestroy*. *winmng_t* object is attached to a device context and this device context must not be manipulated by user code other than via painter callbacks (see *MGL_wmSetWindowPainter*) or via DC obtained from *MGL_wmBeginPaint*.

Members

<i>globalEventHandlers</i>	Table of event handlers that are used prior to window specific ones
<i>capturedEvents</i>	Captured events redirection table
<i>globalCursor</i>	Currently selected global cursor or NULL
<i>rootWnd</i>	The root window that is parent of all other windows
<i>activeWnd</i>	The window under mouse pointer
<i>dc</i>	Device context associated with the manager
<i>invalidatedRegion</i>	Area of device context that needs repainting
<i>dcAttrs</i>	Pointer to next capture entry in the chain
<i>updatingDC</i>	true if inside <i>MGL_wmUpdateDC</i> , false otherwise

A

arc_coords_t, 773
 attributes_t, 774

B

bitmap_t, 776
 bltfx_t, 130, 131, 432, 434, 449, 450, 574, 576, 583, 584,
 587, 588, 725, 778

C

captureentry_t, 780
 codepage_entry_t, 781
 codepage_t, 782
 color_t, 719, 746, 790
 color16_cursor_t, 783
 color256_cursor_t, 785
 colorRGB_cursor_t, 788
 colorRGBA_cursor_t, 787
 CPU_getProcessorName, 25, 26, 27, 28, 29, 30, 31, 32
 CPU_getProcessorSpeed, 26, 27, 28, 29, 30, 31, 32
 CPU_getProcessorSpeedInHZ, 27
 CPU_getProcessorType, 25, 26, 27, 28, 29, 30, 31, 32,
 697
 CPU_have3DNow, 29, 30, 31, 32
 CPU_haveMMX, 25, 26, 27, 28, 29, 30, 31, 32
 CPU_haveRDTSC, 31
 CPU_haveSSE, 29, 30, 32
 CPU_largeInteger, 695
 CPU_processorType, 696
 cursor_t, 669, 791

D

demo, 692

E

event_t, 34, 39, 626, 647, 649, 650, 704, 706, 792
 EVT_allowLEDS, 33
 EVT_asciiCode, 34, 49, 50, 700, 792
 EVT_asciiCodesType, 698
 EVT_eventJoyAxisType, 701
 EVT_eventJoyMaskType, 702, 792
 EVT_eventMaskType, 703, 706
 EVT_eventModMaskType, 704, 793
 EVT_eventMouseMaskType, 705, 792
 EVT_eventType, 39, 46, 706, 792
 EVT_flush, 35, 39, 40, 46, 48
 EVT_getCodePage, 36, 51

EVT_getHeartBeatCallback, 37, 52
 EVT_getMousePos, 38, 53
 EVT_getNext, 35, 37, 39, 40, 46, 47, 48, 52, 54, 470
 EVT_halt, 35, 39, 40, 46, 48, 647, 664
 EVT_isKeyDown, 41
 EVT_joyIsPresent, 42, 44, 45, 47
 EVT_joySetCenter, 42, 43, 44, 47
 EVT_joySetLowerRight, 42, 43, 44, 45, 47
 EVT_joySetUpperLeft, 43, 44, 45, 47
 EVT_masksType, 707
 EVT_peekNext, 35, 37, 39, 40, 46, 47, 48, 52, 54
 EVT_pollJoystick, 47
 EVT_post, 48
 EVT_repeatCount, 34, 49, 50, 792
 EVT_scanCode, 34, 50, 710, 792
 EVT_scanCodesType, 50, 708
 EVT_setCodePage, 36, 51
 EVT_setHeartBeatCallback, 37, 52
 EVT_setMousePos, 38, 53
 EVT_setUserEventFilter, 54

F

fileio_t, 527, 794
 fix32_t, 795
 font_info_t, 201, 796
 font_lib_t, 797
 font_t, 798, 804
 fxpoint_t, 799

G

globalevententry_t, 800
 GM_chooseMode, 55
 GM_cleanup, 56, 68, 69, 70
 GM_driverOptions, 76, 712
 GM_exit, 57, 68
 GM_findMode, 58
 GM_getDoDraw, 59
 GM_getExitMainLoop, 60
 GM_getHaveWin95, 61
 GM_getHaveWinNT, 62
 GM_init, 63, 64, 65, 66, 68, 73, 74, 75, 77, 79, 80, 81,
 85, 86, 87, 88, 89, 90, 91, 92, 94, 96, 711, 712, 713
 GM_initPath, 64
 GM_initSysPalNoStatic, 65, 74
 GM_initWindowPos, 66
 GM_mainLoop, 56, 57, 67, 69, 70, 75, 78, 79, 80, 81, 82,
 83, 90, 91, 92
 GM_modeFlagsType, 713
 GM_modeInfo, 714
 GM_processEvents, 56, 67, 68, 69, 70
 GM_processEventsWin, 56, 67, 68, 69, 70

GM_realizePalette, 71, 93
 GM_registerEventProc, 67, 72, 73
 GM_registerMainWindow, 73
 GM_setAppActivate, 74
 GM_setDrawFunc, 63, 68, 75, 80
 GM_setDriverOptions, 63, 76, 712
 GM_setEventFunc, 78
 GM_setExitFunc, 79
 GM_setGameLogicFunc, 63, 68, 80
 GM_setKeyDownFunc, 63, 81, 82, 83
 GM_setKeyRepeatFunc, 81, 82, 83
 GM_setKeyUpFunc, 81, 82, 83
 GM_setLeftBuffer, 84, 95, 100
 GM_setMode, 63, 85, 89, 94, 97, 714, 715
 GM_setModeExt, 85, 86
 GM_setModeFilterFunc, 88
 GM_setModeSwitchFunc, 85, 87, 89, 94
 GM_setMouseDownFunc, 90, 91, 92
 GM_setMouseMoveFunc, 90, 91, 92
 GM_setMouseUpFunc, 90, 91, 92
 GM_setPalette, 71, 93
 GM_setPreModeSwitchFunc, 94
 GM_setRightBuffer, 84, 95, 100
 GM_setSuspendAppCallback, 74, 96
 GM_startOpenGL, 97
 GM_startStereo, 98, 99
 GM_stopStereo, 98, 99
 GM_stretchType, 715
 GM_swapBuffers, 75, 87, 100, 101
 GM_swapDirtyBuffers, 75, 100, 101
 GMDC, 63, 86, 711
 gmode_t, 155, 157, 801

I

icon_t, 803

L

LZTimerCount, 102, 107
 LZTimerCountExt, 102, 103, 105, 107, 109
 LZTimerLap, 104, 109
 LZTimerLapExt, 103, 104, 105, 107, 109
 LZTimerObject, 716
 LZTimerOff, 103, 106, 109
 LZTimerOffExt, 103, 105, 106, 107, 109
 LZTimerOn, 103, 105, 108
 LZTimerOnExt, 103, 105, 107, 108, 109

M

M_int16, 767
 M_int32, 768
 M_int8, 769
 M_uint16, 770
 M_uint32, 771
 M_uint8, 772
 metrics_t, 804
 MGL_addCustomMode, 113, 121, 329, 401, 402, 507
 MGL_availableBitmap, 114, 351, 354
 MGL_availableCursor, 115, 356

MGL_availableFont, 116, 358
 MGL_availableIcon, 117, 361
 MGL_availableJPEG, 118, 364, 367
 MGL_availablePages, 113, 121, 155, 222, 327, 329, 401, 402, 507
 MGL_availablePCX, 119, 370, 373
 MGL_availablePNG, 120, 375, 378
 MGL_backfacing, 122
 MGL_backModes, 235, 512, 723
 MGL_beginDirectAccess, 123, 124, 197, 198, 740
 MGL_beginDirectAccessDC, 123, 124, 197, 198
 MGL_beginPaint, 125, 199
 MGL_beginPixel, 126, 200, 284, 285, 286, 287, 419, 420
 MGL_bitBlt, 127, 129, 130, 132, 133, 135, 136, 139, 147, 149, 157, 159, 160, 164, 184, 186, 308, 567, 569, 580, 582, 583, 585, 660, 810
 MGL_bitBltCoord, 127, 128, 129, 130, 132, 133, 135, 147, 149, 184, 186, 567, 569, 580, 582, 583, 585
 MGL_bitBltFx, 127, 129, 130, 132, 133, 135, 147, 149, 184, 186, 567, 569, 580, 582, 583, 585, 725, 778
 MGL_bitBltFxCoord, 127, 129, 130, 131, 132, 133, 135, 147, 149, 184, 186, 567, 569, 580, 582, 583, 584, 585
 MGL_bitBltFxFlagsType, 131, 432, 434, 724, 778
 MGL_bitBltPatt, 127, 129, 130, 132, 133, 135, 147, 149, 184, 186, 567, 569, 580, 582, 583, 585
 MGL_bitBltPattCoord, 127, 129, 130, 132, 133, 134, 135, 147, 149, 184, 186, 567, 569, 580, 582, 583, 585
 MGL_blendFuncType, 240, 513, 727, 775, 779
 MGL_bufferFlagsType, 718, 730
 MGL_buildMonoMask, 136
 MGL_charWidth, 137, 138, 386
 MGL_charWidth_W, 137, 138
 MGL_checkIdentityPalette, 129, 132, 139, 427, 428, 432, 433, 435, 440, 441, 572, 573, 574, 575, 577, 579, 582, 585
 MGL_clearDevice, 140, 142, 234, 511
 MGL_clearRegion, 141, 146, 150
 MGL_clearViewport, 140, 142, 234, 296, 297, 511, 545, 546, 556, 557
 MGL_closeFontLib, 143, 360, 410
 MGL_COLORS, 721
 MGL_computePixelAddr, 144
 MGL_copyBitmapToBuffer, 145, 151, 153, 445, 446, 448, 449, 450, 451, 452, 454, 455, 456, 586, 587, 588, 590
 MGL_copyIntoRegion, 146, 150
 MGL_copyPage, 127, 129, 130, 132, 133, 135, 147, 149, 184, 186, 567, 569, 580, 582, 583, 585
 MGL_copyPageCoord, 127, 129, 130, 132, 133, 135, 147, 148, 149, 184, 186, 567, 569, 580, 582, 583, 585
 MGL_copyRegion, 141, 146, 150, 179, 225
 MGL_copyToBuffer, 145, 151, 152, 153, 445, 446, 448, 449, 450, 451, 452, 454, 455, 456, 586, 587, 588, 590, 717
 MGL_createBuffer, 152, 153, 159, 169, 730
 MGL_createCustomDC, 154, 158
 MGL_createDisplayDC, 113, 121, 155, 158, 159, 160, 161, 162, 163, 164, 170, 177, 222, 328, 329, 401, 402, 507
 MGL_createMemoryDC, 154, 156, 157, 159, 161, 164, 170
 MGL_createOffscreenDC, 153, 159, 170

MGL_createScrollingDC, 160, 524
 MGL_createStereoDisplayDC, 156, 159, 161, 162, 514, 570, 571, 755
 MGL_createWindowedDC, 164, 170
 MGL_defaultAttributes, 167
 MGL_defaultColor, 168
 MGL_defRect, 165, 166
 MGL_defRectPt, 165, 166
 MGL_destroyBuffer, 153, 169
 MGL_destroyDC, 154, 156, 158, 159, 161, 163, 164, 170
 MGL_diffRegion, 171, 172, 179, 194, 204, 336, 407, 409, 412, 504, 505, 599, 604, 605, 606
 MGL_diffRegionRect, 171, 172
 MGL_disableDriver, 173, 195, 196, 329, 507
 MGL_disjointRect, 174
 MGL_ditherModes, 732
 MGL_divotSize, 175, 176, 253, 254, 458
 MGL_divotSizeCoord, 175, 176
 MGL_doubleBuffer, 177, 563, 593
 MGL_drawGlyph, 178, 399
 MGL_drawRegion, 179
 MGL_drawStr, 180, 181, 182, 183, 241, 291, 292, 308, 548, 551, 552, 596, 597, 598, 619, 624
 MGL_drawStr_W, 180, 181, 182, 183, 552, 597, 598
 MGL_drawStrXY, 180, 181, 182, 183, 552, 600, 601
 MGL_drawStrXY_W, 180, 181, 182, 183, 552
 MGL_dstTransBlt, 127, 129, 130, 132, 133, 135, 147, 149, 184, 186, 567, 569, 580, 582, 583, 585
 MGL_dstTransBltCoord, 127, 129, 130, 132, 133, 135, 147, 149, 184, 185, 186, 567, 569, 580, 582, 583, 585
 MGL_ellipse, 187, 188, 189, 191, 210, 211, 212, 213
 MGL_ellipseArc, 187, 188, 189, 191, 210, 211, 212, 213, 231
 MGL_ellipseArcCoord, 189
 MGL_ellipseArcEngine, 190, 192, 344
 MGL_ellipseCoord, 187, 191
 MGL_ellipseEngine, 190, 192, 344
 MGL_emptyRect, 174, 193
 MGL_emptyRegion, 194, 204
 MGL_enableAllDrivers, 173, 195, 196, 329, 507
 MGL_enableOpenGLDrivers, 195, 196
 MGL_endDirectAccess, 123, 124, 197, 198, 740
 MGL_endDirectAccessDC, 123, 124, 197, 198
 MGL_endPaint, 125, 199
 MGL_endPixel, 126, 200, 284, 285, 286, 287, 419, 420
 MGL_enumerateFonts, 201
 MGL_equalPoint, 202, 203
 MGL_equalRect, 174, 203
 MGL_equalRegion, 194, 203, 204
 MGL_errorMsg, 205, 329, 473, 547, 733
 MGL_errorType, 473, 733
 MGL_exit, 79, 170, 206, 329, 462, 507
 MGL_fadePalette, 207, 492
 MGL_fatalError, 208
 MGL_fclose, 209, 223, 224, 226, 227, 228
 MGL_fillEllipse, 187, 188, 189, 191, 210, 211, 212
 MGL_fillEllipseArc, 187, 188, 189, 191, 210, 211, 212, 213, 231
 MGL_fillEllipseArcCoord, 211, 212
 MGL_fillEllipseCoord, 210, 213
 MGL_fillPolygon, 214, 290, 544
 MGL_fillPolygonCnvx, 215
 MGL_fillPolygonCnvxFX, 215, 216
 MGL_fillPolygonFX, 214, 216, 217
 MGL_fillRect, 142, 219, 220, 221, 308
 MGL_fillRectCoord, 219, 220, 221
 MGL_fillRectPt, 219, 220, 221
 MGL_findMode, 113, 121, 222, 327, 329, 401, 402, 507
 MGL_FixDiv, 110, 111, 112
 MGL_FixMul, 110, 111, 112, 122
 MGL_FixMulDiv, 110, 111, 112
 MGL_fontBlendType, 258, 529, 735, 775
 MGL_fontLibType, 736, 796
 MGL_fontType, 737
 MGL_fopen, 209, 223, 224, 226, 227, 228, 794
 MGL_fread, 209, 223, 224, 226, 227, 228
 MGL_freeRegion, 141, 146, 150, 179, 225, 407
 MGL_fseek, 209, 223, 224, 226, 227, 228
 MGL_ftell, 209, 223, 224, 226, 227, 228
 MGL_fwrite, 209, 223, 224, 226, 227, 228
 MGL_getActivePage, 229, 300, 508, 561
 MGL_getAlphaValue, 230, 509
 MGL_getArcCoords, 231
 MGL_getAspectRatio, 232, 510
 MGL_getAttributes, 167, 233, 472
 MGL_getBackColor, 234, 248, 511, 520
 MGL_getBackMode, 235, 512
 MGL_getBitmapFromDC, 236
 MGL_getBitmapSize, 237, 238, 351, 354
 MGL_getBitmapSizeExt, 238
 MGL_getBitsPerPixel, 239
 MGL_getBlendFunc, 240, 513
 MGL_getCharMetrics, 242, 243, 259, 386, 596, 597, 598
 MGL_getCharMetrics_W, 242, 243, 597, 598
 MGL_getClipRect, 244, 245, 516, 517
 MGL_getClipRectDC, 244, 245
 MGL_getClipRegion, 244, 245, 246, 247, 518, 519
 MGL_getClipRegionDC, 246, 247
 MGL_getColor, 168, 234, 248, 511, 520
 MGL_getCP, 241, 302, 303
 MGL_getCurrentScanLine, 249, 338, 622
 MGL_getDefaultPalette, 167, 250, 275, 523
 MGL_getDisplayStart, 251, 524
 MGL_getDitherMode, 252, 525
 MGL_getDivot, 175, 176, 253, 254, 458
 MGL_getDivotCoord, 253, 254
 MGL_getDotsPerInch, 255
 MGL_getFont, 256
 MGL_getFontAntiAliasPalette, 257, 528
 MGL_getFontBlendMode, 258, 529
 MGL_getFontMetrics, 242, 243, 259, 386, 596, 597, 598
 MGL_getFullScreenWindow, 260
 MGL_getGammaRamp, 261, 530
 MGL_getGlyphHeight, 262, 263
 MGL_getGlyphWidth, 262, 263
 MGL_getHalfTonePalette, 264, 323
 MGL_getHardwareFlags, 265
 MGL_getJPEGSize, 266, 267, 364, 367
 MGL_getJPEGSizeExt, 267
 MGL_getLineStipple, 268, 531
 MGL_getLineStippleCount, 269, 532
 MGL_getLineStyle, 270, 533

MGL_getPalette, 207, 250, 275, 276, 277, 465, 492, 523, 535, 536
MGL_getPaletteEntry, 275, 276, 536
MGL_getPaletteSize, 250, 275, 277
MGL_getPaletteSnowLevel, 278, 537
MGL_getPCXSize, 271, 272, 370, 373
MGL_getPCXSizeExt, 272
MGL_getPenBitmapPattern, 279, 538, 620
MGL_getPenPixmapPattern, 280, 539, 621
MGL_getPenPixmapTransparent, 281, 540
MGL_getPenSize, 282, 541
MGL_getPenStyle, 283, 542
MGL_getPixel, 126, 284, 285
MGL_getPixelCoord, 284, 285
MGL_getPixelCoordFast, 286, 287
MGL_getPixelFast, 286, 287
MGL_getPixelFormat, 157, 239, 288, 413, 414, 613, 614, 615, 616
MGL_getPlaneMask, 289, 543
MGL_getPNGSize, 273, 274, 375, 378
MGL_getPNGSizeExt, 274
MGL_getPolygonType, 290, 544
MGL_getSpaceExtra, 291, 548
MGL_getTextDirection, 292, 551
MGL_getTextJustify, 293, 553
MGL_getTextSettings, 294, 554
MGL_getTextSize, 295, 555
MGL_getViewport, 296, 297, 516, 517, 518, 519, 545, 546, 556, 557
MGL_getViewportDC, 296, 297
MGL_getViewportOrg, 298, 299, 558, 559
MGL_getViewportOrgDC, 298, 299
MGL_getVisualPage, 229, 300, 508, 561
MGL_getWriteMode, 301, 562
MGL_getX, 302, 303
MGL_getY, 302, 303
MGL_glChooseVisual, 97, 304, 305, 319, 720
MGL_glContextFlagsType, 97, 738
MGL_glCreateContext, 97, 304, 305, 306, 310, 311, 313, 319, 720, 738
MGL_glDeleteContext, 306, 313
MGL_glDisableMGLFuncs, 307, 308
MGL_glEnableMGLFuncs, 307, 308
MGL_glEnumerateDrivers, 309, 316, 317
MGL_glGetProcAddress, 310
MGL_glGetVisual, 311
MGL_glHaveHWOpenGL, 312
MGL_glMakeCurrent, 97, 305, 306, 313
MGL_globalToLocal, 321, 322, 380, 381
MGL_globalToLocalDC, 321, 322
MGL_glOpenGLType, 317, 739
MGL_glRealizePalette, 314, 318
MGL_glResizeBuffers, 315
MGL_glSetDriver, 309, 316, 317
MGL_glSetOpenGLType, 309, 316, 317
MGL_glSetPalette, 314, 318
MGL_glSetVisual, 97, 304, 305, 311, 319, 720, 738
MGL_glSwapBuffers, 320, 763
MGL_halfTonePixel, 323
MGL_halfTonePixel555, 324, 325
MGL_halfTonePixel565, 324, 325
MGL_hardwareFlagsType, 265, 740
MGL_haveWidePalette, 326
MGL_init, 113, 121, 206, 222, 327, 328, 350, 353, 356, 358, 360, 361, 363, 367, 369, 372, 375, 377, 401, 402, 462, 474, 506, 507, 515
MGL_insetRect, 330, 408
MGL_isCurrentDC, 331, 384
MGL_isDisplayDC, 332, 333, 334, 335, 337, 339
MGL_isMemoryDC, 332, 333, 334, 335, 339
MGL_isOffscreenDC, 332, 333, 334, 335, 337, 339
MGL_isOverlayDC, 332, 333, 334, 335, 337, 339
MGL_isSimpleRegion, 336, 819
MGL_isStereoDC, 337
MGL_isVSync, 249, 338, 622
MGL_isWindowedDC, 332, 333, 334, 335, 337, 339
MGL_leftTop, 340, 490
MGL_line, 341, 342, 343, 346, 347, 348, 349
MGL_lineCoord, 341, 342, 343, 345, 346, 347, 348, 349, 499
MGL_lineCoordExt, 342, 343, 345
MGL_lineEngine, 190, 192, 344
MGL_lineExt, 345
MGL_lineRel, 241, 346, 347, 348, 349
MGL_lineRelCoord, 346, 347, 348, 349
MGL_lineStyleType, 742, 775
MGL_lineTo, 241, 346, 347, 348, 349
MGL_lineToCoord, 346, 347, 348, 349
MGL_loadBitmap, 64, 114, 236, 237, 350, 352, 354, 365, 371, 376, 428, 429, 431, 433, 435, 436, 437, 438, 441, 442, 444, 459, 493, 573, 575, 577, 579, 607, 776
MGL_loadBitmapExt, 237, 351, 352
MGL_loadBitmapIntoDC, 237, 350, 351, 352, 353, 355, 368, 374, 493
MGL_loadBitmapIntoDCEExt, 354, 355, 379
MGL_loadCursor, 115, 356, 357, 608
MGL_loadCursorExt, 356, 357
MGL_loadFont, 64, 116, 256, 358, 359, 410, 411, 609, 619
MGL_loadFontExt, 359
MGL_loadFontInstance, 360, 410, 411, 609, 610
MGL_loadIcon, 117, 361, 362, 611
MGL_loadIconExt, 361, 362
MGL_loadJPEG, 118, 266, 363, 365, 367
MGL_loadJPEGExt, 364, 365
MGL_loadJPEGIntoDC, 266, 363, 364, 366, 368, 494
MGL_loadJPEGIntoDCEExt, 368
MGL_loadPCX, 271, 369, 371, 373
MGL_loadPCXExt, 370, 371
MGL_loadPCXIntoDC, 271, 369, 370, 372, 374, 495
MGL_loadPCXIntoDCEExt, 374
MGL_loadPNG, 120, 273, 375, 376, 378, 379, 496, 498
MGL_loadPNGExt, 375, 376
MGL_loadPNGIntoDC, 273, 375, 377, 379, 496, 498
MGL_loadPNGIntoDCEExt, 379
MGL_localToGlobal, 321, 322, 380, 381
MGL_localToGlobalDC, 380, 381
MGL_lockBuffer, 153, 382, 612
MGL_lockToFrameRate, 383
MGL_makeCurrentDC, 331, 384, 506, 507
MGL_mapToPalette, 385
MGL_maxCharWidth, 386
MGL_maxColor, 387, 535
MGL_maxPage, 388

MGL_maxx, 389, 390, 391, 392, 564, 565
 MGL_maxxDC, 389, 390
 MGL_maxy, 389, 390, 391, 392, 564, 565
 MGL_maxyDC, 391, 392
 MGL_memcpy, 393, 394, 395
 MGL_memcpyVIRTDEST, 393, 394, 395
 MGL_memcpyVIRTsrc, 393, 394, 395
 MGL_memset, 396, 397, 398
 MGL_memsetl, 396, 397, 398
 MGL_memsetw, 396, 397, 398
 MGL_mirrorGlyph, 178, 399
 MGL_modeDriverName, 400
 MGL_modeFlags, 113, 222, 327, 401, 743
 MGL_modeFlagsType, 401, 743
 MGL_modeResolution, 113, 121, 222, 327, 329, 401, 402, 507
 MGL_moveRel, 241, 346, 347, 348, 349, 403, 404
 MGL_moveRelCoord, 346, 347, 348, 349, 403, 404
 MGL_moveTo, 241, 405, 406
 MGL_moveToCoord, 405, 406
 MGL_newRegion, 141, 146, 150, 179, 225, 407
 MGL_offsetRect, 330, 408
 MGL_offsetRegion, 194, 204, 409
 MGL_openFontLib, 143, 360, 410, 411, 609, 610, 796
 MGL_openFontLibExt, 143, 410, 411
 MGL_optimizeRegion, 412
 MGL_packColor, 288, 413, 414, 415, 416, 475, 511, 520, 522, 613, 614, 615, 616, 790
 MGL_packColorExt, 413, 414, 416
 MGL_packColorFast, 413, 414, 415, 416
 MGL_packColorFastExt, 415, 416
 MGL_palRotateType, 492, 745
 MGL_penStyleType, 283, 499, 542, 746, 775
 MGL_pixel, 126, 417, 418
 MGL_pixelCoord, 417, 418
 MGL_pixelCoordFast, 419, 420
 MGL_pixelFast, 417, 419, 420
 MGL_polygonType, 290, 544, 747, 775
 MGL_polyLine, 421, 422
 MGL_polyPoint, 421, 422
 MGL_ptInRect, 423, 424
 MGL_ptInRectCoord, 424
 MGL_ptInRegion, 194, 204, 425, 426
 MGL_ptInRegionCoord, 194, 204, 425, 426
 MGL_putBitmap, 139, 350, 351, 364, 369, 370, 375, 427, 428, 429, 431, 433, 435, 436, 437, 438, 441, 442, 444, 459, 573, 575, 577, 579, 776
 MGL_putBitmapDstTrans, 428, 429, 431, 433, 435, 436, 437, 438, 441, 442, 444, 459, 573, 575, 577, 579
 MGL_putBitmapDstTransSection, 428, 429, 430, 431, 433, 435, 436, 437, 438, 441, 442, 444, 459, 573, 575, 577, 579
 MGL_putBitmapFx, 428, 429, 431, 432, 433, 435, 436, 437, 438, 441, 442, 444, 459, 573, 574, 575, 577, 579
 MGL_putBitmapFxSection, 428, 429, 431, 433, 434, 435, 436, 437, 438, 441, 442, 444, 459, 573, 575, 577, 579
 MGL_putBitmapMask, 136, 428, 429, 431, 433, 435, 436, 437, 438, 441, 442, 444, 459, 573, 575, 577, 579
 MGL_putBitmapPatt, 428, 429, 431, 433, 435, 436, 437, 438, 441, 442, 444, 459, 573, 575, 577, 579
 MGL_putBitmapPattSection, 428, 429, 431, 433, 435, 436, 437, 438, 441, 442, 444, 459, 573, 575, 577, 579
 MGL_putBitmapSrcTrans, 428, 429, 431, 433, 435, 436, 437, 438, 441, 442, 444, 459, 573, 575, 577, 579
 MGL_putBitmapSrcTransSection, 428, 429, 431, 433, 435, 436, 437, 438, 441, 442, 443, 444, 459, 573, 575, 577, 579
 MGL_putBuffer, 145, 151, 152, 153, 445, 446, 448, 449, 450, 451, 452, 454, 455, 456, 586, 587, 588, 590, 617, 618, 717
 MGL_putBufferDstTrans, 153, 445, 446, 448, 449, 450, 451, 453, 454, 455, 456, 586, 587, 589, 590
 MGL_putBufferDstTransSection, 445, 446, 447, 448, 449, 450, 451, 453, 454, 455, 457, 586, 587, 589, 590
 MGL_putBufferFx, 445, 446, 448, 449, 450, 451, 453, 454, 455, 457, 586, 587, 589, 590
 MGL_putBufferFxSection, 445, 446, 448, 449, 450, 451, 453, 454, 455, 457, 586, 587, 589, 590
 MGL_putBufferPatt, 445, 446, 448, 449, 450, 451, 453, 454, 455, 457, 586, 587, 589, 590
 MGL_putBufferPattSection, 445, 446, 448, 449, 450, 451, 452, 453, 454, 455, 457, 586, 587, 589, 590
 MGL_putBufferSection, 445, 446, 448, 449, 450, 451, 452, 454, 455, 456, 586, 587, 588, 590
 MGL_putBufferSrcTrans, 152, 153, 445, 446, 448, 449, 450, 451, 453, 454, 455, 456, 586, 587, 589, 590
 MGL_putBufferSrcTransSection, 445, 446, 448, 449, 450, 451, 453, 454, 455, 456, 586, 587, 589, 590
 MGL_putDivot, 175, 176, 253, 254, 458
 MGL_putIcon, 361, 428, 429, 431, 433, 435, 436, 437, 439, 441, 442, 444, 459, 573, 575, 577, 579
 MGL_putMonoImage, 460
 MGL_quickInit, 329, 461
 MGL_random, 463, 464, 566
 MGL_randoml, 463, 464, 566
 MGL_realColor, 156, 157, 465, 475, 521
 MGL_realizePalette, 207, 264, 314, 466, 492, 535, 536
 MGL_rect, 219, 220, 221, 467, 468, 469
 MGL_rectCoord, 467, 468, 469
 MGL_rectPt, 467, 468, 469
 MGL_refreshRateType, 748
 MGL_registerEventProc, 260, 470
 MGL_registerFullScreenWindow, 471
 MGL_restoreAttributes, 167, 233, 472
 MGL_result, 159, 170, 205, 236, 328, 329, 350, 354, 356, 358, 360, 361, 364, 367, 369, 372, 375, 377, 410, 473, 493, 494, 495, 496, 497, 507, 547, 733
 MGL_resume, 474, 592
 MGL_rgbColor, 475, 522
 MGL_rgnEllipse, 476, 477, 486
 MGL_rgnEllipseArc, 476, 477, 478
 MGL_rgnGetArcCoords, 477, 478, 486
 MGL_rgnLine, 479, 480
 MGL_rgnLineCoord, 479, 480
 MGL_rgnPolygon, 481, 482, 483, 484
 MGL_rgnPolygonCnvx, 482, 483
 MGL_rgnPolygonCnvxFX, 481, 483, 484
 MGL_rgnPolygonFX, 481, 484
 MGL_rgnSolidEllipse, 485
 MGL_rgnSolidEllipseArc, 485, 486

MGL_rgnSolidRect, 487, 488, 489
MGL_rgnSolidRectCoord, 487, 488, 489
MGL_rgnSolidRectPt, 487, 488, 489
MGL_rightBottom, 340, 490
MGL_rop3CodesType, 133, 134, 135, 437, 438, 451, 452, 749
MGL_rotateGlyph, 178, 399, 491
MGL_rotatePalette, 207, 492
MGL_saveBitmapFromDC, 236, 351, 354, 493
MGL_saveJPEGFromDC, 364, 367, 494
MGL_savePCXFromDC, 370, 373, 495
MGL_savePNGFromDC, 375, 378, 496
MGL_savePNGFromDCExt, 496, 497
MGL_scanLine, 499
MGL_sectRect, 174, 500, 501, 502, 503, 602, 603
MGL_sectRectCoord, 500, 501, 502
MGL_sectRectFast, 500, 502
MGL_sectRectFastCoord, 501, 503
MGL_sectRegion, 171, 172, 179, 194, 204, 336, 407, 409, 412, 504, 505, 599, 604, 605, 606
MGL_sectRegionRect, 504, 505
MGL_selectDisplayDevice, 328, 329, 506
MGL_setActivePage, 162, 163, 229, 300, 508, 561, 593, 755
MGL_setAlphaValue, 230, 240, 509, 513, 728
MGL_setAspectRatio, 232, 510
MGL_setBackColor, 234, 248, 511, 520
MGL_setBackMode, 235, 512
MGL_setBlendFunc, 240, 513, 727
MGL_setBlueCodeIndex, 163, 514
MGL_setBufSize, 515
MGL_setClipRect, 244, 245, 246, 247, 296, 297, 516, 517, 518, 519, 545, 546, 556, 557
MGL_setClipRectDC, 516, 517
MGL_setClipRegion, 246, 247, 516, 517, 518, 519
MGL_setClipRegionDC, 518, 519
MGL_setColor, 168, 234, 248, 511, 520, 521, 522, 728
MGL_setColorCI, 156, 157, 465, 475, 521, 522
MGL_setColorRGB, 465, 475, 521, 522
MGL_setDefaultPalette, 523
MGL_setDisplayStart, 160, 251, 524, 560, 561
MGL_setDitherMode, 252, 525
MGL_setDotsPerInch, 526
MGL_setFileIO, 209, 223, 224, 226, 227, 228, 527
MGL_setFontAntiAliasPalette, 257, 258, 528, 529
MGL_setFontBlendMode, 257, 258, 528, 529, 735
MGL_setGammaRamp, 261, 530
MGL_setLineStipple, 268, 269, 270, 531, 532, 533
MGL_setLineStippleCount, 268, 269, 270, 531, 532, 533
MGL_setLineStyle, 268, 269, 270, 531, 532, 533, 742
MGL_setOpenGLFuncs, 534
MGL_setPalette, 129, 132, 139, 207, 250, 264, 275, 276, 318, 385, 427, 433, 435, 441, 465, 466, 492, 523, 535, 536, 537, 573, 575, 577, 579, 582, 585
MGL_setPaletteEntry, 276, 535, 536
MGL_setPaletteSnowLevel, 278, 314, 466, 537
MGL_setPenBitmapPattern, 134, 279, 280, 283, 437, 438, 451, 452, 499, 538, 539, 542, 620, 621
MGL_setPenPixmapPattern, 134, 279, 280, 281, 283, 437, 438, 451, 452, 499, 538, 539, 540, 620, 621
MGL_setPenPixmapTransparent, 281, 540, 746
MGL_setPenSize, 282, 541
MGL_setPenStyle, 279, 280, 283, 538, 539, 542, 620, 621, 746
MGL_setPlaneMask, 289, 543
MGL_setPolygonType, 214, 217, 218, 290, 544, 747
MGL_setRelViewport, 296, 297, 545, 546, 556, 557
MGL_setRelViewportDC, 546
MGL_setResult, 473, 547
MGL_setSpaceExtra, 291, 548
MGL_setSuspendAppCallback, 549
MGL_setTextDirection, 292, 551, 759
MGL_setTextEncoding, 552, 760
MGL_setTextJustify, 293, 553, 762
MGL_setTextSettings, 294, 554
MGL_setTextSize, 295, 555
MGL_setViewport, 296, 297, 298, 299, 516, 517, 518, 519, 545, 546, 556, 557, 558, 559
MGL_setViewportDC, 557
MGL_setViewportOrg, 296, 297, 298, 299, 545, 546, 556, 557, 558, 559
MGL_setViewportOrgDC, 558, 559
MGL_setVisualPage, 229, 300, 314, 466, 508, 524, 560, 763
MGL_setWriteMode, 301, 513, 562
MGL_singleBuffer, 177, 563, 593
MGL_sizeX, 389, 390, 391, 392, 564, 565
MGL_sizeY, 389, 390, 391, 392, 564, 565
MGL_srand, 463, 464, 566
MGL_srcTransBlt, 127, 129, 130, 132, 133, 135, 147, 149, 184, 186, 567, 569, 580, 582, 583, 585
MGL_srcTransBltCoord, 127, 129, 130, 132, 133, 135, 147, 149, 184, 186, 567, 568, 569, 580, 582, 583, 585
MGL_startStereo, 163, 514, 570, 571
MGL_stereoBufType, 755
MGL_stopStereo, 163, 514, 570, 571
MGL_stretchBitmap, 428, 429, 431, 433, 435, 436, 437, 439, 441, 442, 444, 459, 572, 573, 575, 577, 579, 776
MGL_stretchBitmapFx, 428, 429, 431, 433, 435, 436, 437, 439, 441, 442, 444, 459, 573, 574, 575, 577, 579
MGL_stretchBitmapFxSection, 428, 429, 431, 433, 435, 436, 437, 439, 441, 442, 444, 459, 573, 575, 576, 577, 579
MGL_stretchBitmapSection, 428, 429, 431, 433, 435, 436, 437, 439, 441, 442, 444, 459, 573, 575, 577, 578, 579
MGL_stretchBlt, 127, 129, 130, 132, 133, 135, 139, 147, 149, 164, 184, 186, 567, 569, 580, 582, 583, 585
MGL_stretchBltCoord, 127, 129, 130, 132, 133, 135, 147, 149, 184, 186, 567, 569, 580, 581, 582, 583, 585
MGL_stretchBltFx, 127, 129, 130, 132, 133, 135, 147, 149, 184, 186, 567, 569, 580, 582, 583, 585
MGL_stretchBltFxCoord, 127, 129, 130, 132, 133, 135, 147, 149, 184, 186, 567, 569, 580, 582, 583, 584, 585
MGL_stretchBuffer, 152, 153, 445, 446, 448, 449, 450, 451, 453, 454, 455, 457, 586, 587, 589, 590, 717
MGL_stretchBufferFx, 445, 446, 448, 449, 450, 451, 453, 454, 455, 457, 586, 587, 589, 590
MGL_stretchBufferFxSection, 445, 446, 448, 449, 450, 451, 453, 454, 455, 457, 586, 587, 588, 589, 590
MGL_stretchBufferSection, 445, 446, 448, 449, 450, 451, 453, 454, 455, 457, 586, 587, 589, 590
MGL_surfaceAccessFlagsType, 591, 756
MGL_surfaceAccessType, 591

- MGL_suspend, 474, 592
 - MGL_suspendAppCodesType, 757
 - MGL_suspendAppFlagsType, 758
 - MGL_swapBuffers, 177, 314, 466, 508, 561, 563, 593, 763
 - MGL_textBounds, 594, 595
 - MGL_textBounds_W, 594, 595
 - MGL_textDirType, 491, 759, 822
 - MGL_textEncodingType, 552, 760
 - MGL_textHeight, 137, 138, 180, 181, 182, 183, 386, 594, 595, 596, 597, 598
 - MGL_textJustType, 292, 293, 551, 553, 762, 822
 - MGL_textWidth, 137, 180, 181, 182, 183, 386, 594, 595, 596, 597, 598, 600, 601
 - MGL_textWidth_W, 138, 594, 595, 597, 598
 - MGL_traverseRegion, 599
 - MGL_underScoreLocation, 600, 601
 - MGL_underScoreLocation_W, 600, 601
 - MGL_unionRect, 174, 500, 501, 502, 503, 602
 - MGL_unionRectCoord, 602, 603
 - MGL_unionRegion, 171, 172, 179, 194, 204, 336, 407, 409, 412, 504, 505, 599, 604, 605, 606
 - MGL_unionRegionOfs, 604, 605, 606
 - MGL_unionRegionRect, 604, 605, 606
 - MGL_unloadBitmap, 351, 364, 370, 375, 607, 678, 679
 - MGL_unloadCursor, 356, 608
 - MGL_unloadFont, 256, 358, 609, 619
 - MGL_unloadFontInstance, 360, 410, 411, 609, 610
 - MGL_unloadIcon, 361, 611
 - MGL_unlockBuffer, 153, 382, 612
 - MGL_unpackColor, 288, 413, 414, 415, 613, 615, 616, 790
 - MGL_unpackColorExt, 416, 613, 614, 615, 616
 - MGL_unpackColorFast, 613, 614, 615
 - MGL_unpackColorFastExt, 616
 - MGL_updateBufferCache, 153, 445, 446, 448, 449, 450, 451, 452, 454, 455, 456, 586, 587, 588, 590, 617, 618, 730
 - MGL_updateFromBufferCache, 153, 445, 446, 448, 449, 450, 451, 452, 454, 455, 456, 586, 587, 588, 590, 617, 618, 730
 - MGL_useFont, 137, 138, 180, 181, 182, 183, 256, 358, 619, 624
 - MGL_usePenBitmapPattern, 279, 538, 620
 - MGL_usePenPixmapPattern, 280, 539, 621
 - MGL_vecFontEngine, 623
 - MGL_vSync, 249, 338, 622
 - MGL_waitVRTFlagType, 100, 101, 524, 560, 593, 763
 - MGL_WIN_COLORS, 721, 722
 - MGL_wmBeginPaint, 625, 629, 630, 633, 659, 826
 - MGL_wmCaptureEvents, 626, 630, 645, 646, 648, 649, 650, 652, 653, 663
 - MGL_wmCoordGlobalToLocal, 627, 630
 - MGL_wmCoordLocalToGlobal, 628, 630, 635
 - MGL_wmCreate, 629, 630, 631, 634, 826
 - MGL_wmCreateWindow, 629, 630, 632, 637, 654, 658, 659, 823
 - MGL_wmDestroy, 629, 631, 634, 657, 826
 - MGL_wmDestroyWindow, 630, 632, 657
 - MGL_wmEndPaint, 625, 633
 - MGL_wmGetRootWindow, 629, 634, 637
 - MGL_wmGetWindowAtPosition, 630, 635, 662
 - MGL_wmGetWindowFlags, 636, 658
 - MGL_wmGetWindowParent, 630, 637, 654
 - MGL_wmGetWindowUserData, 638, 661
 - MGL_wmInvalidateRect, 639, 640, 641, 642, 643, 664
 - MGL_wmInvalidateRegion, 639, 640, 641, 642, 643, 664
 - MGL_wmInvalidateWindow, 630, 639, 640, 641, 642, 643, 659, 664
 - MGL_wmInvalidateWindowRect, 639, 640, 641, 642, 643, 664
 - MGL_wmInvalidateWindowRegion, 639, 640, 641, 642, 643, 664
 - MGL_wmLowerWindow, 630, 644, 651
 - MGL_wmPopGlobalEventHandler, 626, 645, 646, 648, 649, 650, 652, 653, 663
 - MGL_wmPopWindowEventHandler, 626, 630, 645, 646, 648, 649, 650, 652, 653, 663
 - MGL_wmProcessEvent, 626, 629, 645, 646, 647, 649, 650, 652, 653, 663, 664
 - MGL_wmPushGlobalEventHandler, 626, 645, 646, 647, 648, 649, 650, 652, 653, 663
 - MGL_wmPushWindowEventHandler, 626, 630, 645, 646, 648, 649, 650, 652, 653, 663
 - MGL_wmRaiseWindow, 630, 644, 651, 664
 - MGL_wmRemoveGlobalEventHandler, 626, 652, 663
 - MGL_wmRemoveWindowEventHandler, 626, 645, 646, 648, 649, 650, 652, 653, 663
 - MGL_wmReparentWindow, 630, 654
 - MGL_wmSetGlobalCursor, 655, 656
 - MGL_wmSetWindowCursor, 630, 655, 656
 - MGL_wmSetWindowDestructor, 630, 632, 657, 661
 - MGL_wmSetWindowFlags, 630, 636, 644, 651, 658, 659, 660, 764
 - MGL_wmSetWindowPainter, 625, 629, 630, 633, 659, 664, 823, 826
 - MGL_wmSetWindowPosition, 627, 628, 630, 635, 660, 664
 - MGL_wmSetWindowUserData, 630, 638, 657, 659, 661
 - MGL_wmShowWindow, 630, 659, 662, 664
 - MGL_wmUncaptureEvents, 626, 630, 645, 646, 648, 649, 650, 652, 653, 663
 - MGL_wmUpdateDC, 625, 629, 633, 639, 640, 641, 642, 643, 647, 659, 660, 664, 826
 - MGL_wmWindowFlags, 658, 764, 824
 - MGL_writeModeType, 129, 148, 301, 428, 441, 445, 562, 573, 579, 765, 775
 - MGLBUF, 717
 - MGLDC, 155, 157, 719, 801
 - MGLVisual, 720
 - mono_cursor_t, 805
 - MS_getPos, 665, 667
 - MS_hide, 666, 668, 672
 - MS_moveTo, 665, 667
 - MS_obscure, 666, 668, 672
 - MS_setCursor, 356, 669
 - MS_setCursorColor, 670
 - MS_setCursorColorExt, 671
 - MS_show, 666, 668, 672
- P**
- palette_ext_t, 806

palette_t, 207, 719, 807
 pattern_t, 808
 pixel_format_t, 157, 790, 809
 pixpattern_t, 816
 pixpattern16_t, 812
 pixpattern24_t, 813
 pixpattern32_t, 814
 pixpattern8_t, 815
 point_t, 817

R

rect_t, 818
 region_t, 819

S

segment_t, 820
 span_t, 821
 SPR_destroyBitmap, 673, 678, 679
 SPR_draw, 674, 675, 678, 679
 SPR_drawExt, 674, 675, 676, 677
 SPR_drawSection, 676
 SPR_drawSectionExt, 677
 SPR_mgrAddOpaqueBitmap, 673, 674, 675, 676, 677, 678, 679
 SPR_mgrAddTransparentBitmap, 673, 674, 675, 676, 677, 678, 679
 SPR_mgrEmpty, 678, 679, 680, 681
 SPR_mgrExit, 681, 682

SPR_mgrInit, 674, 675, 676, 677, 680, 681, 682

T

text_settings_t, 822

U

ULZElapsedTime, 683, 684, 688, 689
 ULZReadTime, 683, 684, 688, 689
 ULZTimerCount, 685, 686, 687, 688, 689
 ULZTimerLap, 685, 686, 687, 688
 ULZTimerOff, 685, 686, 687, 688
 ULZTimerOn, 685, 686, 687, 688
 ULZTimerResolution, 683, 684, 685, 688, 689

W

window_t, 657, 661, 823, 825
 windowevententry_t, 825
 winmng_t, 629, 800, 826
 writeCursor, 693

Z

ZTimerInit, 690
 ZTimerInitExt, 690, 691